UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

Henrique Machado Gasparotto

DESENVOLVIMENTO DE ARQUITETURA MULTI-TENANT PARA INTERNET DAS COISAS

SANTA MARIA, RS
2016

Henrique Machado Gasparotto

DESENVOLVIMENTO DE ARQUITETURA MULTI-TENANT PARA INTERNET DAS COISAS

Trabalho de graduação apresentado ao curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do grau de **Engenheiro de Computação**.

Orientador: Prof. Dr. Carlos Henrique Barriquello

SANTA MARIA, RS
2016

Ficha gerada com os dados fornecidos pelo autor

Gasparotto, Henrique Machado
Desenvolvimento de arquitetura Multi-Tenant para Internet das Coisas / Henrique Machado Gasparotto.-2016.
121 p.; 30cm

Orientador: Prof. Dr. Carlos Henrique Barriquello
Trabalho de graduação - Universidade Federal de Santa Maria, Centro de Tecnologia, Curso de Engenharia de Computação, Santa Maria, RS, 2016.

1. Multi-Tenant 2. Arquitetura de software 3 Internet das Coisas I. Barriquello, Carlos Henrique II. Gasparotto, Henrique Machado III. Desenvolvimento de arquitetura Multi-Tenant para Internet das Coisas.

Henrique Machado Gasparotto

DESENVOLVIMENTO DE ARQUITETURA MULTI-TENANT PARA INTERNET DAS COISAS

Trabalho de graduação apresentado ao curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM), como requisito parcial para a obtenção do grau de **Engenheiro de Computação**.

Aprovado em __ de _____ de 2016:

_____
Carlos Henrique Barriquello, Dr. (UFSM)
(Orientador)

_____
Componente da Banca 1 (UFSM)

_____
Componente da Banca 2 (UFSM)

SANTA MARIA, RS
2016

AGRADECIMENTOS

À minha família e amigos pelo apoio e incentivo durante minha trajetória no curso.

Ao professor Carlos Henrique Barriquello pelas dicas, paciência e orientação durante o desenvolvimento do projeto, sem as quais o mesmo não seria possível.

A todos os professores pelo conhecimento transmitido durante o curso.

Ao CNPq, ao Governo Federal e à UFSM pela oportunidade de aumentar meus conhecimentos em uma instituição de ensino canadense.

Ao meu pai, Jorge Luiz Gasparotto, pela ideia que deu início ao projeto.

À minha mãe, Suzimara Machado Gasparotto, pelas dicas e incentivo no desenvolvimento do projeto.

À minha noiva, Marina Vaucher Sampaio, pelo apoio, incentivo e auxílio no desenvolvimento deste projeto.

A todos que contribuíram de forma direta ou indireta para a conclusão do trabalho.

“Nosso futuro é escrito pelas nossas ações no presente” – autor desconhecido

RESUMO

DESENVOLVIMENTO DE ARQUITETURA MULTI-TENANT PARA INTERNET DAS COISAS

AUTOR: Henrique Machado Gasparotto
ORIENTADOR: Prof. Dr. Carlos Henrique Barriquello

A arquitetura Multi-Tenant vem crescendo nos últimos anos e se consolidando como a principal alternativa para criação de SaaS – *Software as a Service*. O presente projeto visa utilizar essa alternativa de arquitetura de software para a criação de uma aplicação genérica visando outro campo em crescimento – o da Internet das Coisas (IoT - *Internet of Things*). O projeto cria uma solução completa de software, utilizando o .NET Framework e algumas das principais tecnologias de desenvolvimento web do momento, para controlar as "coisas" em uma cidade: Santa Maria, RS, é a cidade utilizada como exemplo. Os dados utilizados são fictícios. O foco do projeto é a criação da arquitetura Multi-Tenant; assim, a aplicação criada para demonstração é bastante simplificada, trazendo dois elementos, apenas, ou "coisas": medidores de nível de lixo em containers e controladores de iluminação pública, com indicação de ligado/desligado. O projeto faz uso de uma arquitetura de dados *Single Database, Separated Schema* para armazenamento dos dados dos Tenants individualmente, além de uma separação em camadas que segue os mais modernos modelos de desenvolvimento de software. Projetos futuros envolvem a evolução desse trabalho de forma a auxiliar na criação de "cidades do futuro" no presente.

Palavras-chave: Multi-Tenant. Arquitetura de software. Internet das coisas. .NET.

ABSTRACT

DEVELOPMENT OF MULTI-TENANT ARCHITECTURE FOR INTERNET OF THINGS

AUTHOR: Henrique Machado Gasparotto
ADVISOR: Prof. Dr. Carlos Henrique Barriquello

The Multi-Tenant architecture has been growing in the last few years and consolidating itself as the main alternative to the creation of SaaS – Software as a Service. The present project aims to utilize this alternative in software architecture to the creation of a generic application aiming another growing field – IoT (Internet of Things). The project creates a complete software solution, using .NET Framework and some of the main web development technologies of the moment, in order to control the "things" in a city: Santa Maria, RS was used in this example. The data used in this project is fake. The focus here is the creation of a Multi-Tenant architecture; being so, the application created is overly simplified, bringing two elements, only, or "things": a trash container level measurer and public illumination controllers, with on/off indication. To save this data while guaranteeing the individual storage of Tenant data, the project uses a Single Database, Separated Schema data architecture, together with and layering separation of concerns that follows the latest patterns in software industry. Future projects involve the evolution of this work in order to help creating the "cities of the future" in the present.

Keywords: Multi-Tenant. Software architecture. Internet of Things. .NET.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AJAX	*Assynchronous JavaScript And XML*
EF	Entity Framework
JSON	*JavaScript Object Notation*
MVC	*Model-View-Controller*
ORM	*Object-Relational Mapping* (Mapeamento Objeto-Relacional)
OWIN	*Open Web Interface for .NET*
SaaS	*Software as a Service* (Software como Serviço)
SOA	*Service Oriented Architecture* (Arquitetura Orientada a Serviços)
SQL	*Structured Query Language*
POCO	*Plain Old Class Object*
IoT	*Internet of Things* (Internet das Coisas)
POO	Programação Orientada a Objetos
URL	*Uniform Resource Locator*

SUMÁRIO

1 INTRODUÇÃO

O conceito de IoT tem crescido muito nos últimos tempos. Por tratar-se de um conceito que visa adicionar alguma espécie de inteligência artificial para melhorar o dia-a-dia de todos, é uma área que atrai muita atenção de pesquisadores e desenvolvedores. Nesse contexto entram as plataformas de IoT, que visam permitir ao usuário tratar os dados retornados por cada um dos dispositivos, aqui tratados como "coisas" (*things*). Esse trabalho visa justamente isso: fazendo utilização de conceitos de software avançados e amplamente discutidos, criar a arquitetura para uma plataforma de IoT genérica.

O conceito de SaaS tenta colocar um fim ao tradicional "software de prateleira", e tem tido uma grande aceitação por parte dos consumidores para diversas aplicações. Além disso, traz como um dos pontos principais a capacidade de criar soluções mais próximas do negócio do cliente, criando situações personalizadas, algo difícil de ser atingido em aplicações tradicionais.

Dentro desse conceito, é necessário que exista uma arquitetura robusta, capaz de suportar múltiplos acessos e de criar elementos únicos para os usuários aproveitando um núcleo comum. É aí que entra o conceito da arquitetura Multi-Tenant. Os "Tenants" são os usuários, então o nome explica bem do que se trata.

É esse "trio" de tecnologias/padrões que fazem parte desse projeto: a criação de uma arquitetura de software Multi-Tenant escalável, confiável, segura e customizável, baseando-se em conceitos de SaaS e com dados (fictícios) de IoT. Os 4 pilares das aplicações Multi-Tenant são a base para todas as ações executadas dentro do projeto.

A arquitetura Multi-Tenant contrasta com a arquitetura Multi-Instance. Na primeira, temos uma única instância da aplicação servindo múltiplos usuários. Na segunda, temos uma instância da aplicação por usuário. Antes de definirmos qual das duas utilizaremos, é importante que tenhamos uma noção do que precisamos em nossa aplicação. Aplicações Multi-Instance servem melhor aplicações que precisam de configuração de baixo-nível (configurações referentes ao SO, por exemplo) por usuário, além de serem menos complexas que as primeiras. Já as aplicações Multi-Tenant funcionam para clientes que tem uma infraestrutura de hardware similar, e acabam sendo mais complexos por precisarem evitar questões como *data leakage* entre os diferentes Tenants.

Existem diversas formas de abordagem para uma aplicação Multi-Tenant, todas elas com seus prós e contras. Ao longo desse trabalho, foram feitas algumas considerações e escolhas, que ficarão claras neste documento. A implementação feita é de uma arquitetura que permita que tenhamos confiabilidade, escalabilidade, segurança e customização em um nível aceitável para cada Tenant.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ARQUITETURA MULTI-TENANT

A arquitetura Multi-Tenant é aquela em que múltiplos clientes/consumidores são consolidados no mesmo serviço. Por isso o fato de ser tão utilizada em aplicações do tipo SaaS. De acordo com [9], essa técnica teve seu ponto de partida a partir do website "salesforce.com". Essa técnica permite a utilização de uma "piscina" de recursos, o que melhora a utilização da aplicação ao eliminar a necessidade de prover cada consumidor com o máximo de sua carga de trabalho.

Ainda de acordo com [9], um sistema Multi-Tenant confiável precisa ser capaz de fazer duas ações em termos de escalabilidade: *scale up* e *scale out*. O primeiro diz respeito a consolidar vários consumidores em um único servidor até atingir seu limite. A partir daí o *scale out* deve entrar em ação, para garantir a utilização de outros servidores quando necessário, pois não é efetivo em termos de custos realizar o *scale up* indefinidamente.

2.1.1 MULTI-TENANCY

O conceito de *Multi-Tenancy* é algo que vem crescendo no meio da arquitetura de software. De acordo com [2], trata-se de uma alternativa ao conceito de *multi-deployment*, que seria algo como várias instâncias da mesma aplicação sendo instaladas em diferentes máquinas. Esse conceito, com o advento da nuvem, está se tornando obsoleto. É muito complicado uma empresa ser realmente global precisando instalar a sua aplicação em cada cliente para garantir o seu acesso. Assim, o conceito de *Multi-Tenancy* se encaixa bem com outro conceito, o de SaaS (*Software as a Service*), e acaba facilitando para o desenvolvedor essa globalização.

Para que o conceito fique mais claro, é importante dissociar o *tenant* do *user*. São dois conceitos que podem acabar se confundindo dentro dessa arquitetura. Conforme comenta [3], "*Multitenancy*, ou *Multi-Tenancy*, é a referência a um modo de operação de software onde múltiplas instâncias independentes de uma ou múltiplas aplicações operam em um ambiente compartilhado". Logo, dependendo da forma como desenvolvemos a aplicação, teremos um *tenant* com mais de um usuário. O *tenant* pode ser classificado como sendo a empresa, organização ou indivíduo que utilizará o software.

Por tratar-se de uma arquitetura compartilhada, existem alguns conceitos que precisam ser levados em consideração. De acordo com [1], o mais importante requerimento desse tipo é o **isolamento**. Os *tenants* individuais não podem ter suas tarefas afetadas pelas atividades de outros. Além disso, conceitos como **disponibilidade**, **escalabilidade**, **custo**, **nível de customização** e também **questões legais**, para garantir que a aplicação não viola nenhuma lei local, no caso de *tenants* em diferentes países ou regiões.

As questões da disponibilidade e escalabilidade merecem um parágrafo à parte. Ainda de acordo com [1], é preciso levarmos em consideração esses dois aspectos que vão afetar diretamente a experiência do usuário em nossa aplicação. Os *tenants* precisam que a aplicação esteja disponível 24/7. Para isso, não basta apenas que o serviço na nuvem contratado faça sua parte: é preciso que a aplicação seja estruturada de forma

a garantir que não haverá erros em momentos cruciais da aplicação. Isso se torna ainda mais importante quando a aplicação lida diretamente com o negócio dos clientes. Já a questão de escalabilidade é um pouco mais complexa. É possível que cada *tenant* tenha necessidades diferentes em termos de armazenamento de dados, requisições, armazenamento e memória. É preciso que a aplicação se adapte às necessidades de cada um, garantindo que a aplicação funcione com desempenho ao menos satisfatório.

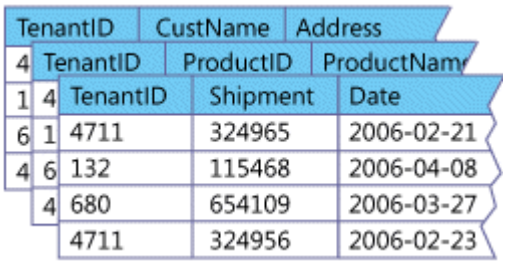
2.2 ARQUITETURA DE DADOS

De acordo com [9], uma base de dados multi-tenant terá, geralmente, um *schema* base que especifica todos os padrões de dados da aplicação. E a base de dados multi-tenant deve manter uma instância desse *schema* base para cada Tenant. Essa mesma referência também faz alusão a três métodos de separação da arquitetura de dados: *shared machine*, *shared process* e *shared table*. Como o artigo em questão trata do ponto de vista do serviço, essa nomenclatura não será utilizada. A nomenclatura tratada em [1] é mais literal e deixa mais claro o que o projeto utilizará.

2.2.1 *Shared Database, Shared Schema*

A **Figura 2.1** traz a alternativa de trabalhar em uma única tabela e *schema* para os Tenants. Nota-se que os dados são separados através de um "TenantId", que é o identificador único de cada um deles. Também é possível notar-se que não há uma separação lógica entre os dados, e isso só será feito quando os dados forem acessados de fato.

Figura 2.1 – *Shared Database, Shared Schema*



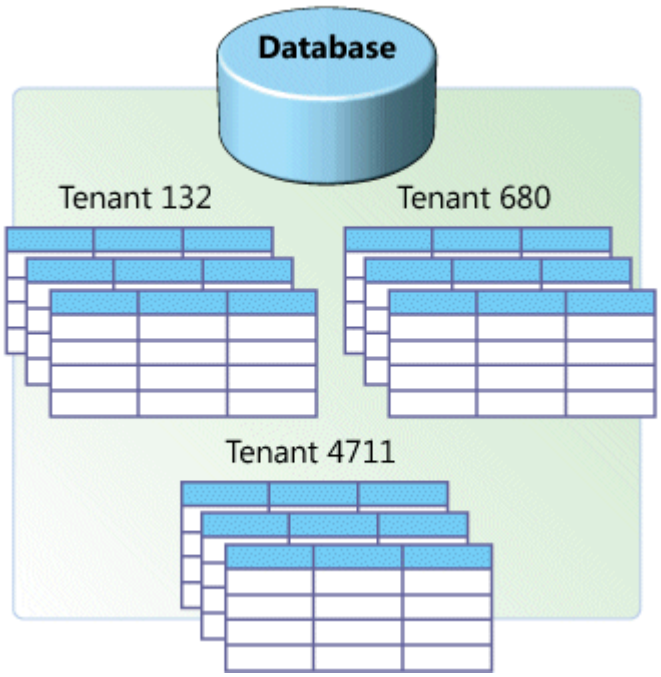
Fonte: [10]

De acordo com [10], essa abordagem possui os menores custos de hardware e backup, pois todos os Tenants são servidos na mesma base de dados. Além disso, também fornece o maior número possível de Tenants por servidor de dados. Entretanto, como a segurança é precária, para evitar *data leakage* acidental ou malicioso, se faz necessário um desenvolvimento à prova de falhas na parte de segurança da aplicação.

2.2.2 *Shared Database, Separated Schema*

Essa abordagem fornece um meio termo: quando o Tenant se subscreve para o sistema, seu *schema* é criado em cima da base de dados utilizada. Para que a escalabilidade seja válida nesse caso, é importante levar em consideração o conceito de *scale out* mencionado anteriormente. Essa abordagem possui um limite interessante, apontado em [9]: o armazenamento necessário é linear (ou perto disso) em todos os casos. Já memória varia bastante de acordo com o banco de dados utilizado. O que se pode notar é que, acima de um determinado número de instâncias do *schema*, o espaço em disco começa a ser um impeditivo. Em [9], esse valor é o de 10 mil instâncias

Figura 2.2 – *Shared Database, Separated Schema*

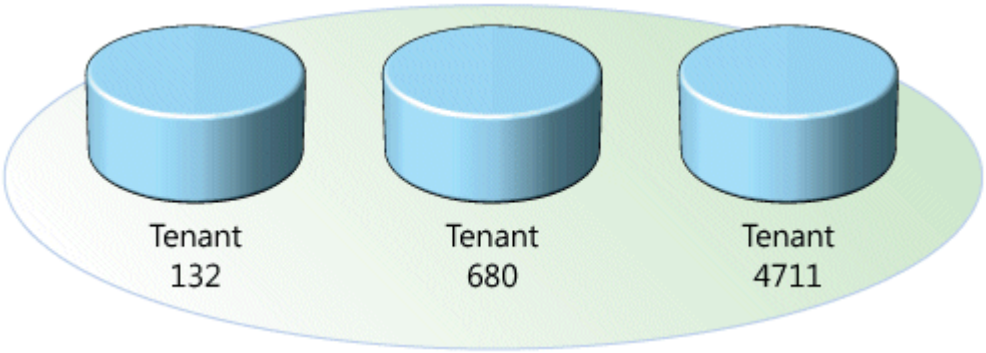


Fonte: [10]

2.2.3 *Isolated Database*

Nessa abordagem, os recursos e o código da aplicação são geralmente compartilhados entre os diferentes Tenants, mas os dados são guardados de formas distintas. Eles ficam logicamente isolados, cada um em seu conjunto, ou base de dados. Cada Tenant possui metadados que o associam a sua base de dados, evitando acesso a dados incorretos ou mesmo *data leakage*.

Figura 2.3 – *Isolated Database*



Fonte: [10]

De acordo com [10], essa abordagem traz uma série de vantagens e alguns outros problemas. É facilmente customizável, podendo ser atribuída a diferentes necessidades do cliente, inclusive com adaptações específicas. Além disso, backup de dados ou mesmo correção em caso de erros no servidor de dados é uma tarefa relativamente simples. Entretanto, essa abordagem também traz alguns problemas de ordem financeira: é muito mais custoso, em todos os sentidos, manter uma base de dados por Tenant. Gasta-se mais com manutenção, hospedagem etc., e o número de Tenants da aplicação é limitado pelo número de bases de dados que podem ser adquiridas pelo serviço.

2.3 ARQUITETURA ORIENTADA A SERVIÇOS (SOA)

A arquitetura de software orientada a serviços tem sido uma das mais utilizadas atualmente. Com a necessidade de criar soluções que funcionem em uma gama grande de sistemas operacionais e dispositivos com muitas diferenças entre si, é interessante que tenhamos vários serviços que vão sendo consumidos por esses diferentes dispositivos. Por exemplo, temos uma aplicação que terá um cliente web e outros em Google Android, Apple iOS e Windows Phone. É inviável que tenhamos apenas uma biblioteca de classes para atender a esses 4 clientes, uma vez que cada um espera uma biblioteca em um formato. Assim, a solução é criar uma camada de serviços que podem ser consumidos por cada um dos clientes, garantindo a mesma experiência para qualquer usuário.

A definição de SOA é bem ampla: “Um conjunto de componentes que podem ser invocados e cujas descrições de interface podem ser publicadas e descobertas”, conforme [7]. Outras definições são aceitas, mais específicas, como a também apresentada em [7]: “As políticas, práticas e frameworks que habilitam a funcionalidade da aplicação a ser provida e consumida como um conjunto de serviços publicado com uma

granularidade relevante para o consumidor do serviço". Essa última definição é mais decisiva para nossas pretensões nesse trabalho, uma vez que indica o que precisamos fazer em termos de nossa SOA: criar funcionalidades que serão consumidas com uma granularidade relevante – leia-se, webservice.

A utilização de webservices é bastante óbvia. Ainda segundo [7], a orientação em direção aos serviços começou a partir de 2001, mas é somente agora que a SOA está tomando forma devido à quantidade de dispositivos diferentes disponíveis. Assim, a utilização de uma arquitetura desse tipo se faz necessária em vários tipos de aplicações. Outro ponto a favor desse tipo de arquitetura é que tira "peso" do cliente. Em outras palavras, a aplicação cliente fica muito menor, ocupando menos espaço em disco, já que todos os serviços por ela acessados estarão na nuvem.

2.4 INTERNET DAS COISAS (IOT)

A Internet das Coisas é um conceito que tem chamado a atenção do mundo do desenvolvimento nos últimos anos. De fato, temos um projeto extremamente interessante desenvolvido por estudantes do Inatel (Instituto Nacional de Telecomunicações). O Taurus é uma plataforma de IoT Brasileira. De acordo com [11], a ideia foi construir um painel de controle que oferecesse a possibilidade de controlar a maioria dos dispositivos de IoT. Para isso, o trabalho faz uso de atuadores, analógicos e/ou digitais, e/ou sensores de diversos tipos, englobando praticamente todos os dispositivos de IoT que estão sendo disponibilizados atualmente.

A iniciativa destacada em [11] não é nova, de fato. Existem outras plataformas com fins similares. Por exemplo, a Taurus foi baseada no DeviceHub.net. Temos ainda plataformas na nuvem que podem ser adaptadas para a Internet das Coisas, como o Microsoft Azure, a AWS (*Amazon Web Services*) e o PubNub.

Mas, afinal, o que é a Internet das Coisas. De acordo com [12], IoT é um paradigma no qual objetos inteligentes colaboram de forma ativa com outros objetos físicos e virtuais disponíveis na Internet. Os ambientes de IoT são caracterizados por um alto grau de heterogeneidade de dispositivos e protocolos de rede. Assim, é tarefa de cada plataforma de IoT saber lidar com esses diferentes equipamentos e transformar os sinais digitais e analógicos transmitidos pelos dispositivos em informação com a qual o usuário possa lidar e alterar, se necessário.

Ainda de acordo com [12], um dos grandes problemas na concepção da IoT é a falta de padronização existente na área. Em geral, diferentes plataformas de *middleware* adotam diferentes modelos de programação, em geral não compatíveis entre si, o que prejudica o desenvolvedor do produto final. Além disso, as soluções não abordam requisitos de escalabilidade de forma apropriada, podendo prejudicar uma plataforma de IoT que se desenvolva. Por fim, é preciso lidar com a quantidade massiva de dados que é transmitida pelos dispositivos interconectados. Para isso, mecanismos devem ser adicionados de forma a manipulá-los de forma eficiente, sempre atendendo às necessidades de aplicações.

Em [13], tem-se uma discussão muito interessante a respeito de IoT e como ela deve ser concebida. O artigo fala em *cloud centric Internet of Things*, ou seja, uma IoT totalmente centrada na nuvem. Outra opção é a *Thing centric*, ou centrada na "coisa". O principal foco quando se fala em *cloud centric* é o

envolvimento de serviços da web sendo o grande foco da aplicação, enquanto as “coisas” contribuem apenas com os dados. Esses dados são enviados e então tratados pelos serviços.

Ainda em [13], podemos ver que a proposta de IoT *cloud centric* é uma arquitetura flexível e aberta, que é centrada no usuário e simplesmente utiliza as “coisas” como dados a serem trabalhados. Esse tipo de abordagem é muito comum atualmente, com os conceitos de *Data Mining*, por exemplo. Alguns desafios se abrem a partir dessa abordagem, como questões de privacidade, análise de dados, visualização dos dados, que fogem um pouco dos problemas técnicos da construção de tais sistemas.

2.5 MAPEAMENTO OBJETO-RELACIONAL (ORM)

De acordo com [14], Mapeamento Objeto-Relacional, ou ORM, é uma técnica para converter dados entre sistemas de tipos incompatíveis em Linguagens de Programação Orientadas a Objetos. Isso cria uma “base de dados virtual de objetos” que pode ser utilizada de dentro da linguagem de programação. Em outras palavras, é uma camada adicional de abstração que faz o mapeamento entre os objetos (classes da aplicação) e os relacionamentos e entidades (base de dados).

Esse tipo de abordagem, ainda de acordo com [14], elimina a necessidade de mapeamento manual entre os objetos e o SQL. Isso facilita o desenvolvimento e permite que os desenvolvedores foquem no que realmente importa: a regra de negócio da aplicação e como aplica-la de forma inteligente ao usuário. Existem algumas diferenças entre a classe e a tabela, que precisam ser controladas e trabalhadas, e isso pode ser visto na **Tabela 2.1**.

Tabela 2.1 – Comparação entre os elementos do objeto e relacionais

O objeto	**Relação**
Transiente	Persistente
Classes	Tabelas
• *Herança / polimorfismo*	
Atributos/propriedades	Colunas
Objetos	Linhas
• Atributos de instância	• Campos
• *ID implícito: referência*	• *ID explícito: chave primária*

Fonte: [14] (traduzido)

A **Tabela 2.1** compara os elementos e podemos notar como eles se equivalem no mapeamento. As grandes diferenças e principais dificuldades para o mapeamento, de acordo com [14], estão nos itens em itálico. Não há um meio simples de lidar com a herança e o polimorfismo inerentes às linguagens orientadas a objetos e nem como fazer os relacionamentos entre os identificadores funcionarem de forma simples. Cada ORM utiliza uma forma para esse mapeamento.

2.5.1 Entity Framework

O Entity Framework, conhecido como EF, introduziu o conceito de ORM (*Object Relational Mapping*) no .NET e Visual Studio. De acordo com [6], o conceito central desse ORM é o Entity Data Model, um modelo conceitual que mapeia os modelos do domínio da aplicação para o esquema da base de dados. O EF é responsável por fazer a ponte entre o modelo de domínio e a base de dados, normalmente utilizando o banco de dados SQL Server, evitando que o desenvolvedor precise se preocupar com queries SQL, que adicionariam complexidade à aplicação.

Antes do Entity Framework Code First, a mais nova (e melhor) das versões do EF, tínhamos outras duas alternativas: Database-First ou Model-First. Como os nomes sugerem, tínhamos que criar uma base de dados ou um modelo dessa base de dados para podermos criar o código necessário pelo EF. Tais abordagens não foram bem aceitas na comunidade de desenvolvimento por não serem muito práticas, uma vez que mais adicionavam complexidade do que auxiliavam na utilização do EF. Já o Code First, como o nome sugere, utiliza a abordagem da criação do código antes de qualquer coisa. O modelo de domínio é definido no código e então mapeado em tabelas da base de dados.

De acordo com [6], ainda, esses três modelos de execução são considerados um *workflow*, ou fluxo de trabalho. Isso porque Database-First, Model-First e Code First são um conjunto de passos a serem realizados para a criação de uma base de dados baseada no EF. Esses passos podem ser feitos de forma automatizada, em alguns casos, o que faz com que o EF se torne uma opção cada vez melhor. Atualmente, o EF está em sua versão 6.

2.6 DESENVOLVIMENTO DE SOFTWARE EM CAMADAS

O desenvolvimento de software em camadas é um dos principais padrões de projeto de software atualmente. É muito complicado, para não dizer impossível, desenvolver uma aplicação corporativa com diversos módulos sem uma estrutura em camadas. O código fica completamente ilegível e de dificílima manutenção e entendimento. É aí que entra o desenvolvimento em camadas, que permite a separação entre os elementos, ou *Separation of Concerns*.

De acordo com [8], temos algumas camadas comuns em todo o desenvolvimento de software. Isso não quer dizer que todas elas são necessárias em todos os aplicativos, mas é uma base sobre a qual o desenvolvedor precisa trabalhar e adaptar para sua necessidade. As camadas são:

- *Business Logic Layer* (BLL): A camada de negócios da aplicação, que contém todas as classes do negócio, bem como os serviços que as acessam e adicionam comportamentos às mesmas. Alguns padrões de projeto são utilizados nessa camada, como o Domain Model.
- *Service Layer* (SL): Essa camada é, muitas vezes, tratada como uma subcamada da BLL. Explica-se: os serviços são normalmente utilizados para acessar e/ou adicionar comportamento às classes do domínio. Ela fornece a ponte entre a camada de apresentação e a camada de negócios, normalmente transformando o *model* em um *viewmodel*.

- *Data Access Layer* (DAL): A DAL é a camada que fornece a abstração para acesso aos dados. A base de dados está embaixo dela, e a função da DAL é evitar que as camadas superiores conheçam a implementação da mesma. Em outras palavras, se o formato dos dados for alterado, a única camada que o será é a DAL. Aqui, estão presentes padrões como o *Repository* e o *Unit of Work*.
- *Application Layer*: A camada de aplicação fornece uma abstração entre os serviços que estão sendo mostrados na SL e como eles serão utilizados na camada de apresentação. A ideia é evitar que a apresentação saiba exatamente como são implementados os serviços e por isso esse interfaceamento é tão importante.
- *Presentation Layer*: A camada de apresentação é a camada que contém as views que o usuário irá enxergar. Essa camada utiliza os serviços das camadas inferiores para acesso às funcionalidades da aplicação e as expõe para o cliente.
- *User Experience Layer* (UUX): A camada de experiência de usuário é a menos utilizada destas. Muitas vezes confundida com a anterior, é muito importante em aplicações onde o usuário precisa de uma experiência única, diferente do habitual. Aqui, é implementada a lógica para criar esses detalhes diferentes do usual e para adicionar dinamicidade à camada de apresentação.

As camadas nem sempre são utilizadas dessa forma. Ainda de acordo com [8], a camada de serviços pode ser abreviada em uma só, fundida ou com a BLL ou com a camada de aplicação. A camada de UUX também é comumente evitada, incorporando-se à camada de apresentação sem prejuízo nas funcionalidades. Essa não separação é indicada em casos em que a complexidade da camada seria tão pequena que invalida o conceito de camadas, causando mais mal do que bem.

2.7 SEGURANÇA

O conceito de segurança é extremamente amplo, e não é diferente quando falamos de segurança de software. De acordo com [22], "Segurança de software" é o processo de identificar e expurgar problemas no software em si. Em outras palavras, o conceito tenta construir software que possa evitar ataques maliciosos de forma proativa.

Ainda de acordo com [22], nota-se que o problema de segurança é grande, e vai desde bugs a defeitos e problemas de concepção. Por isso é tão importante seguir alguns parâmetros durante o desenvolvimento do software, pois isso irá garantir que é à prova de falhas que possam prejudicar o funcionamento e a segurança do mesmo.

O livro, ainda, faz uma comparação entre dois conceitos: bugs e defeitos. Um defeito nada mais é do que vulnerabilidades decorrentes da implementação ou do design do mesmo. Um defeito pode permanecer escondido durante anos até ser descoberto, trazendo grandes e problemáticas consequências. Já um bug é um problema no software a nível de implementação. Um bug é um problema de codificação que pode ser

encontrado e resolvido de forma relativamente fácil. Logo, o bug será resolvido antes mesmo que o aplicativo chegue às mãos do cliente. O que faz do defeito algo muito mais importante de ser levado em consideração.

[22] traz, ainda, alguns elementos que são úteis na hora de testes um sistema procurando defeitos: os testes de estresse. O que o teste faz é testar de forma massiva o software, imitando sua utilização em nível máximo. Isso faz com que o software seja testado em seus limites e traz alguns resultados interessantes: o primeiro deles, óbvio, é o conhecimento do limite do software, fazendo com que o mesmo não seja mais executado a partir daquele limite; o outro, menos óbvio e mais raro como resultado, é encontrar um problema que poderia potencialmente paralisar o sistema, causando prejuízos incalculáveis.

A segurança também envolve ataques maliciosos e, consequentemente, formas de proteger dados contra eles. Alguns conceitos são utilizados e não estão dissociados um do outro, como a criptografia e a autenticação e consequente autorização de usuários a partir de credenciais fornecidas.

2.7.1 Criptografia MD5

A criptografia é um campo de estudos muito amplo, com um objetivo claro: transformar uma cadeia de bytes legível em algo completamente fora de contexto para o caso de algum ser malicioso interceptar a mensagem. O início da criptografia não está claro: alguns dizem que começou no Império Romano, como uma forma dos césares enviarem mensagens seguras, e outros falam na Segunda Guerra Mundial como o marco, através das máquinas Enigma do regime nazista. Entretanto, qualquer que seja o resultado histórico dessa busca, tem-se uma questão clara: desde sempre o ser humano busca esconder mensagens potencialmente perigosas ou secretas. E, na era da comunicação digital, é preciso estar cada vez mais atento a isso.

Pensando nisso, existe um algoritmo de autenticação conhecido como MD5. Ele é utilizado em uma gama grande de pesquisas e aplicações por sua robustez e virtual impossibilidade de ser quebrado. De acordo com [15], o algoritmo MD5 é uma derivação do DES (*Data Encryption Standard*), tornado mais seguro para os dias atuais.

2.7.2 Autenticação e Autorização

O artigo [16] faz uma comparação entre os diferentes métodos de autenticação e autorização disponíveis atualmente. Dessa comparação é possível notar-se que as diferentes aplicações necessitam de diferentes formas de autenticação. Por exemplo, a validação por VPN é somente utilizada para criação de uma rede privada para comunicação entre duas máquinas. Para aplicações, a autenticação mais comum e segura ainda é a utilização de nome de usuário e senha. E, claro, adicionando criptografia no armazenamento de senhas.

Já o artigo [17] traz uma visão um pouco diferente e mais voltada à implementação. A grande sacada deste é a amostragem de soluções existentes para evitar o desenvolvimento de algo totalmente novo, o que causaria muitos problemas e um atraso no desenvolvimento. Além disso, não há nenhuma garantia de que

essa solução desenvolvida seria, de fato, segura. Uma das soluções prontas para autenticação e autorização é o OWIN.

2.7.2.1 OWIN

O OWIN, de acordo com [18], define um padrão de interfaceamento entre servidores .NET e suas aplicações. O objetivo do OWIN é separar completamente servidor e aplicação, facilitando o desenvolvimento de módulos mais simples para desenvolvimento .NET para web e, por ser um padrão aberto, estimular o desenvolvimento do ecossistema de soluções open source para desenvolvimento web com o .NET Framework.

2.8 .NET FRAMEWORK

O .NET Framework é um framework desenvolvido pela Microsoft para desenvolvimento de software. De acordo com [19], foi desenvolvido baseado na linguagem de programação C#, que também é a linguagem de programação preferida para desenvolvimento de aplicações nessa plataforma. As aplicações .NET executam em um ambiente de software, e não diretamente na máquina, sendo traduzidas para essa CLR, ou *Common Language Runtime*, e interpretadas em tempo de execução.

2.8.1 Generics

De acordo com [19], Generics é um conceito introduzido no C# 2.0 e que se tornou muito útil e robusto com a evolução da linguagem ao longo dos anos. A ideia é muito simples: criar elementos genéricos, que possam ter seu tipo definido em tempo de desenvolvimento e que possam ser replicados para quantos tipos for necessário. O tipo de dados é definido na classe através de um parâmetro, normalmente precedido pelo letra T.

Generics é muito útil porque permite a aplicação de padrões de design sem muita codificação. Em aplicações corporativas com muitos módulos, a presença de módulos genéricos costuma poupar horas, dias e até meses de desenvolvimento. Por exemplo, podemos criar módulos “RepositórioEntidade1”, “RepositórioEntidade2” e “RepositórioEntidade3”, implementando-os um a um, individualmente. Ou podemos criar um módulo genérico “RepositorioEntidade<Número>”, desenvolvê-lo apenas uma vez utilizando o parâmetro genérico Número e então atribuir esse parâmetro às três entidades. Outra vantagem é que o código fica centralizado, facilitando a manutenção.

2.8.2 ASP.NET MVC

O desenvolvimento de aplicações web dentro do .NET Framework possui, basicamente, duas vias: o ASP.NET Web Forms, clássico, que foi a primeira implementação que a Microsoft disponibilizou; e o

ASP.NET MVC Framework, que é considerado por muitos como uma abordagem mais moderna para aplicações escaláveis, de acordo com [2]. Por isso, a escolha em nosso trabalho é pelo segundo.

A sigla MVC, *Model-View-Controller*, representa um padrão de arquitetura de software. Esse padrão tem sido muito utilizado, especialmente em aplicações web, pois separa a representação da informação da interação do usuário com ele. De acordo com [4], o M (*Model*) consiste nos dados da aplicação, regras de negócio, lógica e funções. Já o V (*View*) é o que o usuário da aplicação estará enxergando em sua tela. Por fim, o C (*Controller*) é a entidade que faz a conexão entre os dois anteriores: ele precisa ser capaz de entender tanto modelo como visão para realizar essa comunicação.

2.8.2.1 MVC Framework

Já o MVC Framework é a implementação da Microsoft desse padrão, atualmente em sua versão 6. Esse framework é a base utilizada nas aplicações ASP.NET MVC. Ele se baseia, atualmente, em um *view engine* chamado Razor, bastante poderoso, que permite inclusive a adição de códigos C# diretamente na *view*, as chamadas *Razor Expressions*. Ele utiliza uma sintaxe similar a PHP e ao ASP clássico para adicionar esse tipo de código tipicamente do "lado do servidor" nas páginas do cliente. Além disso, conforme comenta [5], o Razor possui alguns métodos auxiliares, os *Helpers*, que são extremamente úteis na criação de formulários e outros elementos dentro das *views*.

2.8.2.2 Razor

O ASP.NET MVC introduziu o conceito de *view engines*, ou motores de view, no desenvolvimento de aplicações web. Um desses motores é o Razor. De acordo com [20], ele permite a inserção de conteúdo dinâmico nas páginas web, alterando o HTML de acordo com alguns parâmetros presentes na linguagem. Em outras palavras, permite a utilização de conceitos da linguagem C#, como condicionais e loops, para adição de conteúdo dinâmico ao HTML. Esse conteúdo é então traduzido, o HTML, gerado, e este, enviado para o cliente.

O Razor, ainda, permite um conceito muito importante no desenvolvimento ASP.NET MVC: as views fortemente tipadas. Através da diretiva "@model", é possível definir um tipo para a view. Esse tipo é normalmente utilizado para realizar o binding entre o controller e a view, para passagem de dados durante métodos HTTP como POST. Ele também é muito útil para amostragem de dados nas views.

2.9 MICROSOFT AZURE

Até agora, falamos muito em nuvem, sem explicitar muito bem o que seria isso. O Microsoft Azure é um modelo de IaaS – *Infrastructure as a Service* – que está hospedado na nuvem. Ele pode ser classificado dentro do modelo de nuvem privada. É esse serviço que utilizaremos para hospedagem de nossa arquitetura multi-tenant.

A escolha pelo Azure se dá por alguns fatores: como comenta [1], podemos escolher entre várias abordagens. Essas abordagens são: bases de dados diferentes por usuário, mesma base de dados com esquema diferente por usuário e mesma base de dados, mesmo esquema com identificador de *tenant*. A escolha por cada uma dessas abordagens depende de outros fatores dentro da aplicação que será desenvolvida, mas o mais importante aqui é vermos que o Azure suporta tais possibilidades.

2.10 BING MAPS

De acordo com [21], o Bing Maps fornece uma plataforma com diversas opções de API para as aplicações desenvolvidas que necessitam de mapas. Uma delas é o V8 Web Control, um dos controles de mapeamento e serviços de geolocalização mais completos e universais do mercado. É desenvolvido em JavaScript, o que faz dele um padrão para desenvolvimento não só na web, mas em muitas aplicações para dispositivos móveis também.

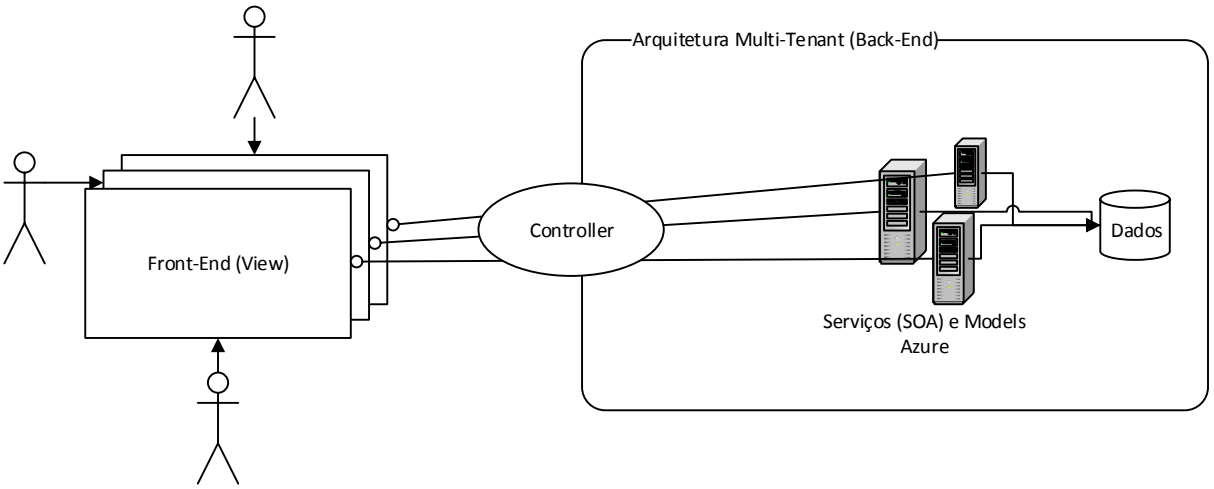
O V8 Web Control, ainda de acordo com [21], permite o desenvolvimento em uma outra linguagem de scripts, o TypeScript. Essa linguagem nada mais que uma implementação do JavaScript para o .NET Framework, o que faz com que seja mais parecida com o C# do que com o Java. A sua utilização não é muito comum, uma vez que a Microsoft não conseguiu alterar o padrão estabelecido com o JavaScript e suas muitas bibliotecas (como jQuery, Angular JS, Node.js etc.), então acaba não sendo muito relevante esta informação.

O controle fornece muitas funcionalidades além da simples amostragem de um mapa. O mapa é capaz de entender diversos eventos (que podem ser tratados via código), além de trazer possibilidades de desenho, adição de *pushpins* e amostragem da rua em si, com o StreetSide do Bing Maps.

3 ARQUITETURA DE SOFTWARE DO PROJETO

A arquitetura de software é o grande foco do projeto. Conforme já foi discutido, foi criada uma arquitetura Multi-Tenant para utilização na Internet das Coisas (IoT). Essa arquitetura fala por si própria, e pode ser vista em detalhes na **Figura 3.1**.

Figura 3.1 – Arquitetura de software do projeto



Fonte: documento de apresentação do projeto

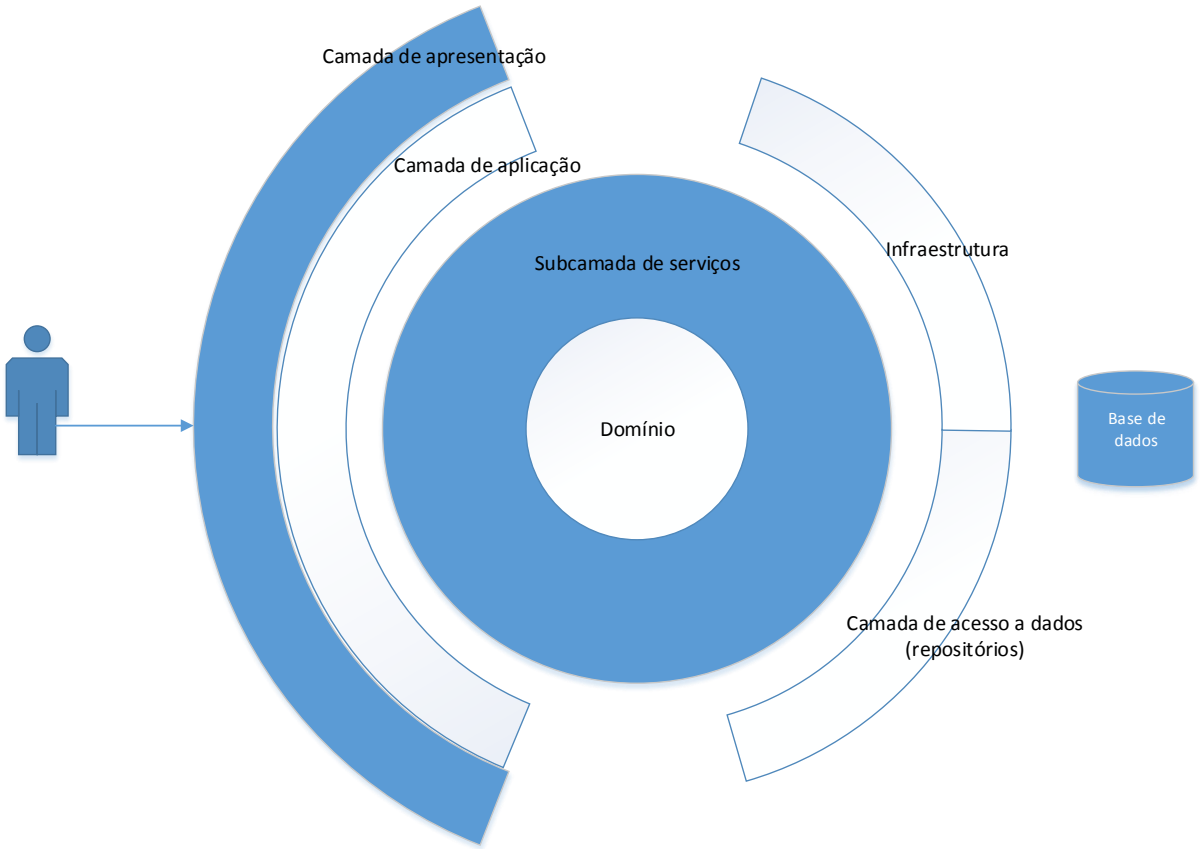
Com a arquitetura mostrada, podemos entender que há uma conexão muito clara entre os Tenants e suas próprias views, que representam o front-end. Cada Tenant possui suas views, de forma separada. Os dados são armazenados em serviços hospedados no Microsoft Azure e acessados através de um (ou vários) Controller (s) comum.

A **Figura 3.1** mostra claramente o que o conceito de *Multi-Tenancy* se propõe a fazer: separar os dados de cada Tenant, garantindo segurança, escalabilidade, confiabilidade e customização.

3.1 DESENVOLVIMENTO DO PROJETO EM CAMADAS

O desenvolvimento de software em camadas é uma das questões mais importantes em grandes aplicações. Ele garante um conceito conhecido como *Separation of Concerns*, onde cada camada tem sua função e não ultrapassa essa função por nada. Nesse tipo de arquitetura, caso uma camada necessite de algo que não possui, ela irá conversar com as demais e encontrar essas informações. A **Figura 3.2** mostra como as camadas estão divididas no projeto.

Figura 3.2 – Estrutura em camadas do projeto



Fonte: próprio autor (embasamento na Figure 3-3 de [8])

Para garantir a confiabilidade do projeto desenvolvido, foi necessária a utilização desse padrão. Dessa forma, garante-se que possíveis problemas com o banco de dados sejam facilmente solucionados sem alterações bruscas de código. Além disso, também se encontra problemas muito mais facilmente.

A escalabilidade do projeto também é garantida com esse tipo de escolha: serviços funcionam muito melhor para lidar com múltiplas requisições do que um repositório de dados seguindo o padrão *Repository*, por exemplo. Logo, a adição das camadas de serviço e de aplicação faz-se necessária para garantir o funcionamento da aplicação para 1 ou N Tenants, com N tendendo ao infinito.

A segurança também é afetada por essa escolha de desenvolvimento em camadas. Explica-se: com a separação entre as funções de cada camada, é muito mais difícil para um invasor atingir o que realmente importa, que são os dados da aplicação. Para o fazer, é necessário passar pelas camadas de apresentação, aplicação, serviços e acesso a dados. Logo, o invasor tem seu trabalho dificultado também pela estrutura do projeto criado.

3.1.1 Camada de Domínio

A camada de domínio, ou de negócios, contém as interfaces e classes necessárias aos demais projetos. É nessa camada que estão as classes POCO que representam as entidades do banco de dados.

3.1.1.1 Subcamada de serviços

Em constante comunicação com a camada de domínio, esses serviços funcionam como uma ponte para os dados da camada de domínio. Em outras palavras, são esses serviços que dizem o que as classes de domínio **fazem**, enquanto a camada de domínio indica o que elas **são**.

3.1.2 Camada de Acesso a dados

No projeto, foi implementada com o padrão *Repository*, embora isso não seja uma exigência. A camada de acesso a dados serve para encapsular a lógica de acesso a banco de dados em um só lugar, independentemente do provedor de dados que está funcionando por baixo.

3.1.3 Camada de Aplicação

Faz a ponte entre os serviços do domínio e a apresentação para o usuário. Utiliza serviços que fazem o acesso às camadas inferiorer e retorna esses dados à camada de apresentação.

3.1.4 Camada de Apresentação

Controla o “lado do servidor” da apresentação ao usuário. Basicamente, todas as requisições feitas pelo usuário da aplicação passam por aqui. No projeto criado, também concentra as tarefas da camada de “User Experience”, que conteria o “lado do cliente”. É ela que recebe todas as requisições e as envia para serem cuidadas pela aplicação.

3.2 DEFINIÇÃO DA ARQUITETURA DE DADOS

A definição da arquitetura de dados talvez seja a parte mais importante do projeto, pois é aqui que está sendo definido todo o futuro da aplicação. Algumas considerações precisam ser levadas em conta, baseadas em três arquiteturas possíveis e que foram apresentadas. Também é possível trabalhar com versões híbridas das mesmas, dependendo das necessidades da aplicação. Como o projeto precisa ter somente uma arquitetura de dados, foi feita uma análise, algumas considerações e então a escolha, que será explicada em detalhes.

3.2.1 Considerações

Antes da escolha, algumas considerações foram realizadas. As opções estão demonstradas claramente na **Tabela 3.1**. É importante ressaltar que os valores para número de Tenants não são baseados em testes. São apenas utilizando o bom senso e a bibliografia no que diz respeito a aplicações Multi-Tenant, levando em consideração a máxima de que, quanto mais customização, menos Tenants.

Tabela 3.1 – Comparação entre as arquiteturas de dados possíveis

	Vantagens	Desvantagens	Número de Tenants
Shared Database, Shared Schema	Maior simplicidade, menor custo, fácil manutenção, maior escalabilidade	Não permite customização, maior possibilidade de *data leakage*	Tende ao infinito, dependendo do banco de dados e do servidor utilizados
Shared Database, Separated Schema	Baixo custo, boa escalabilidade, boa customização, média simplicidade, segurança	Não permite uso massivo, implementação não trivial, difícil manutenção	Funciona bem até 10000 Tenants. Depois, tende a sobrecarregar.
Isolated Database	Segurança e customização máximas	Alto custo, difícil manutenção, baixa escalabilidade	Depende dos recursos da empresa

Fonte: próprio autor (baseado nas referências bibliográficas)

Os pontos levados em consideração nessa análise são vários. A ênfase está no número de Tenants (estimativa) permitido, nível de simplicidade, dificuldade da manutenção, nível de customização possível, segurança, escalabilidade e custo. Da **Tabela 3.1**, é possível tirar algumas conclusões:

1. O conceito de *Shared Database, Shared Schema* se assemelha muito ao que vemos com o software de prateleira: não permite qualquer customização dos dados de cada Tenant, pois todos utilizam a mesma base, com a separação entre os dados de cada sendo feita através de uma chave;
2. O conceito de *Shared Database, Separated Schema* une o melhor dos dois mundos: alguma customização, segurança na separação dos dados dos Tenants e baixo custo;
3. O método de *Isolated Database* é indicado apenas em casos em que se tem um negócio, mas ele é tratado de formas diferentes para os diferentes Tenants. É extremamente seguro, uma vez que os Tenants não têm sequer conhecimento das bases de dados dos demais, e permite o máximo de customização, pois as bases de dados não tem relação alguma umas com as outras. O grande problema está no custo: é inviável, para a maior parte das empresas, manter bancos de dados individuais para cada cliente, mesmo na nuvem.

Algo bem utilizado em muitas arquiteturas Multi-Tenant é a junção de alguns desses conceitos. A junção mais comum é a separação dos Tenants em grupos, tendo cada grupo sua base de dados. Assim,

permite-se a customização, tem-se a separação dos dados (seja por diferentes *schemas* ou não) e o custo não é tão elevado.

Cada aplicação possui uma necessidade e a seleção da arquitetura de dados deve seguir o que é melhor para ela. Por isso o planejamento é tão importante, bem como o conhecimento de cada uma das opções disponíveis.

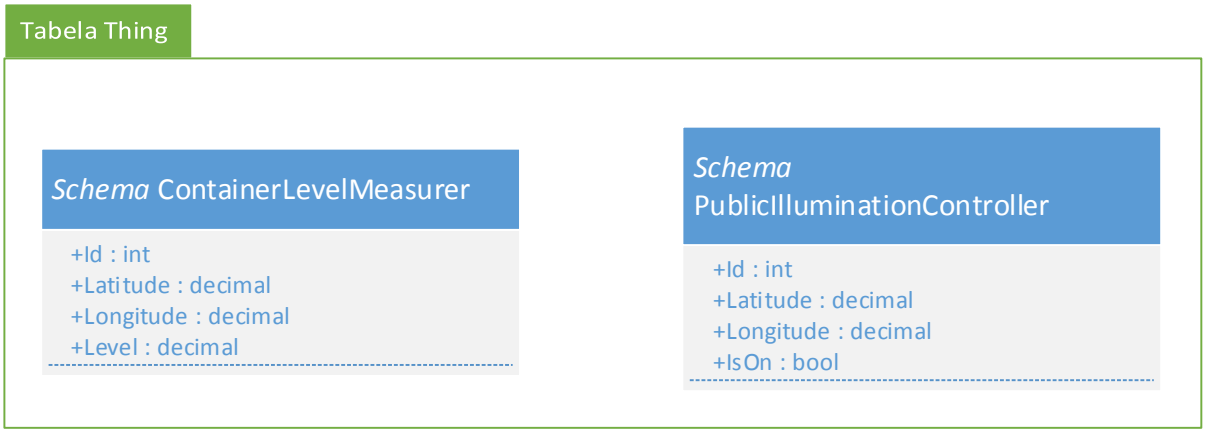
3.2.2 Escolha

A escolha para o projeto é por uma única base de dados com diferentes *schemas*. Essa escolha foi feita pela natureza da aplicação: as cidades estão sendo tratadas como Tenants – logo, não se espera um grande número de Tenants utilizando a mesma. Assim, o número de Tenants e o custo estão de acordo com essa alternativa.

Além disso, com ela também se utilizam as vantagens de customização, permitindo que cada Tenant tenha um grupo de "coisas" como alvo, além da separação dos dados de cada um deles, garantindo que não haverá nenhum tipo de *data leakage*.

A **Figura 3.3** mostra como os dados foram estruturados com essa alternativa. Para esta, dois tipos de "coisas" são levados em consideração: o ContainerLevelMeasurer, medidor de nível de lixo em contêineres, e o PublicIlluminationController, controlador das lâmpadas de iluminação pública. Podemos notar que ambas estão presentes em uma única tabela, Thing, com diferentes *schemas*. São esses *schemas* que serão levados em consideração e utilizados pelos Tenants no projeto.

Figura 3.3 – Única tabela, múltiplos *schemas*



Fonte: próprio autor

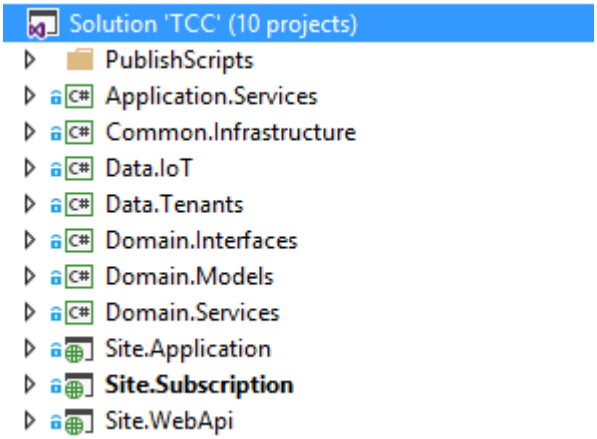
4 DESENVOLVIMENTO

4.1 CRIAÇÃO DA SOLUÇÃO COMPLETA

O projeto foi desenvolvido utilizando o IDE (*Integrated Development Environment* – ou Ambiente de Desenvolvimento Integrado) Microsoft Visual Studio Community 2015. Este é a ferramenta de desenvolvimento mais utilizada para aplicações .NET de qualquer tipo, além de ser considerado por muitos desenvolvedores como o melhor e mais completo ambiente de desenvolvimento do mercado.

Obedecendo aos conceitos de desenvolvimento em camadas, a solução está estruturada como mostra a **Figura 4.1**. Nota-se que cada projeto possui um prefixo, indicando a camada à qual pertence.

Figura 4.1 – Estrutura em camadas da solução



Fonte: Próprio autor

Os projetos mostrados na **Figura 4.1** são dos tipos *ASP.NET Web Application* e *Class Library*.

4.2 CRIAÇÃO DA BASE DE DADOS DOS TENANTS

Um conceito muito comum em aplicações modernas é o do Mapeamento Objeto-Relacional, ou ORM (da sigla, em inglês, *Object-Relational Mapping*). Esse conceito foi utilizado para definição de ambas as bases de dados no projeto. Nesse primeiro momento, foi realizada a criação da base de dados para os Tenants, os clientes da aplicação criada. O ORM que foi utilizado para tal é o Entity Framework 6, a versão mais recente da ferramenta. No momento da escrita deste documento, a versão mais recente deste é a 6.1.3, que foi utilizada.

A base de dados dos Tenants e todas as regras de acesso à mesma foram criados no projeto Data.Tenants, na camada de acesso a dados. Para tanto, foi feita a instalação de alguns pacotes nesse projeto, necessários para adição de referências que foram utilizadas no projeto, utilizando o NuGet através do comando "Install-Package", como a seguir:

```
Install-Package EntityFramework –project Data.Tenants
Install-Package Microsoft.AspNet.Identity.EntityFramework –project
Data.Tenants
Install-Package Microsoft.AspNet.Identity.Core –project Data.Tenants
```

As classes de domínio (modelos) que foram utilizadas para a criação dessa base de dados estão no projeto Domain.Models, como convém ao modelo de desenvolvimento de software em camadas utilizado nesse projeto.

4.2.1 Criação da classe Tenant (POCO)

Utilizando o padrão POCO (*Plain Old Class Object*), a classe Tenant será criada. Essa classe é responsável pela representação dos clientes da aplicação. Podemos ver o código da classe no **Anexo A**. A classe contém apenas 1 (uma) propriedade: *HostName*. Ela serve para guardar o valor da URL que representará a aplicação para cada Tenant.

É interessante notar-se que a classe Tenant utiliza o conceito de herança, muito comum em aplicações .NET. Esse conceito consiste em uma classe herdar de outra, algumas informações, sejam métodos, propriedades, atributos ou mesmo valores constantes. No caso específico, Tenant herda de IdentityUser, classe do namespace "Microsoft.AspNet.Identity.EntityFramework". A maior parte das informações dos Tenants no projeto são comuns à classe IdentityUser. Ela contém propriedades como *UserName*, *Email* e *PasswordHash*, que servem muito bem ao que é desejado em termos de segurança na arquitetura Multi-Tenant. A ação de fazer com que Tenant herde de IdentityUser é uma escolha que também favorece a implementação da política de segurança da arquitetura aqui criada, baseada no Owin.

4.2.2 Criação do TenantDbContext

A classe DbContext é a responsável por centralizar o acesso aos dados no EntityFramework. Normalmente, utiliza-se um DbContext para cada base de dados às quais a aplicação acessa, embora isso não seja uma regra.

Para o acesso aos dados de Tenants, foi criada uma classe, TenantDbContext (**Anexo B**). Essa classe não herda diretamente de DbContext, e sim de uma segunda classe, genérica, chamada IdentityDbContext<T>, onde T é a classe à qual o contexto se refere: Tenant, nesse caso. A utilização dessa classe genérica se dá pelo mesmo motivo pelo qual derivamos nossos Tenants de IdentityUser: isso favorece a implementação da política de segurança com o Owin.

Além dos elementos herdados de IdentityDbContext<Tenant>, foi criado um método estático, Create(), que cria e retorna um objeto do tipo TenantDbContext. A criação desse método estático evita o acesso desnecessário ao construtor da classe.

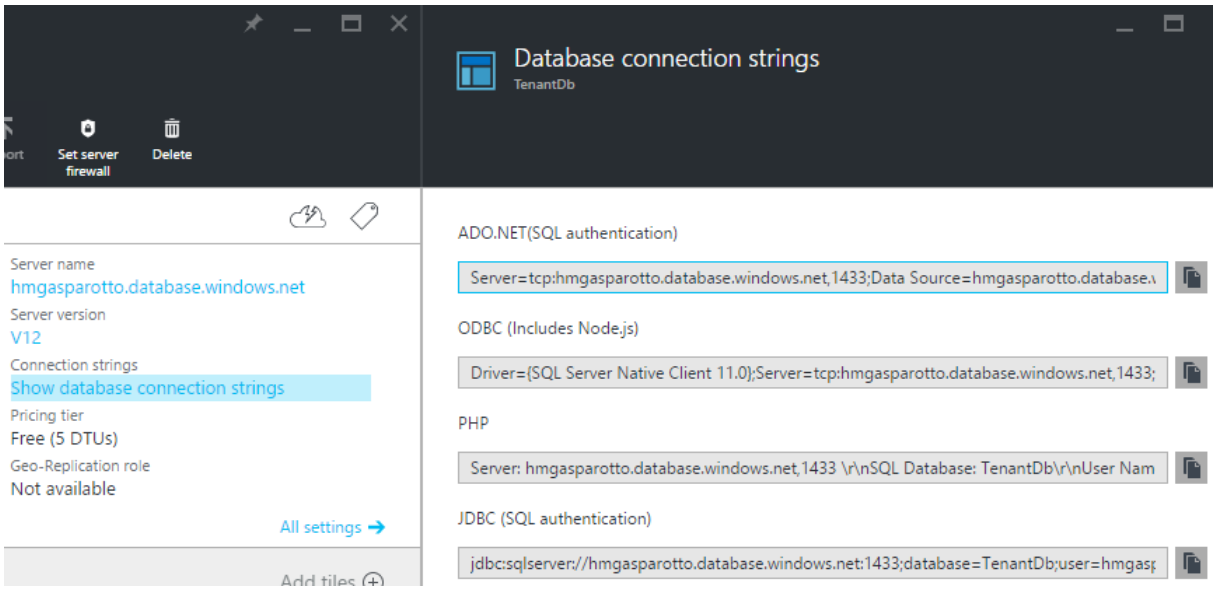
TenantDbContext é a representação da base de dados de Tenants. Ainda que não possua explicitamente, a classe herda de IdentityDbContext<Tenant> um DbSet<Tenants>, ou seja, uma coleção de dados que representa a tabela Tenants.

4.2.3 Criação da SQL Database no Azure

A base de dados foi criada no Microsoft Azure. Para tanto, basta, no *Azure Portal*, selecionar *SQL Databases* e, então, "Add". As configurações básicas foram utilizadas, inclusive para escolha do servidor no qual o banco de dados ficará hospedado.

Após a criação do banco de dados, precisamos obter a string de conexão a ele. Para tanto, foi selecionada essa opção no portal e, como mostra a **Figura 4.2**, obtida a string de conexão. Após a obtenção, a string foi colocada no arquivo App.config, no projeto Data.Tenants, com o nome "TenantDatabase".

Figura 4.2 – Obtenção da string de conexão à SQL Database no Azure



Fonte: Próprio autor

4.2.4 Utilização de Code-First Migrations para inicializar a base de dados

Com os elementos de suporte prontos, foi realizada a criação da base de dados propriamente dita, utilizando o conceito de Migrations do Entity Framework Code First. Para tanto, foi feita a execução de alguns comandos no "Package Manager Console" do Microsoft Visual Studio, como a seguir:

```
Enable-Migrations -project Data.Tenants
Add-Migration InitialCreate -project Data.Tenants
Update-Database -project Data.Tenants
```

O primeiro comando foi executado para habilitar as *Migrations* na base de dados de Tenants. Isso permite um controle sobre todas as alterações na classe Tenant e posterior atualização da base de dados. Já o segundo comando foi executado para criar uma *Migration* – InitialCreate. Após a execução desse comando, nota-se a criação de uma classe no projeto Data.Tenants – essa classe está mostrada no **Anexo C**. Observa-

se que a mesma possui dois métodos básicos: Up() e Down(), que servem para alterar a base de dados utilizando comandos SQL encapsulados em métodos do Entity Framework, como “Create Table”, “Drop Table” e outros. Ao final, o comando “Update-Database”, que executou as alterações presentes na *Migration* InitialCreate, criando a base de dados.

4.3 PREPARAÇÃO DO ACESSO AOS DADOS COM O PADRÃO *REPOSITORY*

Para acesso aos dados, foi utilizado o padrão *Repository*, responsável por encapsular a lógica de acesso a dados, fazendo com o que o cliente não tenha acesso direto à implementação da base de dados.

Esse padrão está previsto nas interfaces do namespace Domain.Interfaces.Repositories.

4.3.1 Criação da interface genérica IRepositoryBase<T>

A interface genérica IRepositoryBase<T> foi criada para ser implementada, direta ou indiretamente, em todos os repositórios de dados do projeto. Ela está presente no **Anexo D**. Pode-se notar por esse código que a mesma prevê métodos para todas as operações de acesso e modificação de dados: *Select* (GetAll() e Find()), *Insert* (Add()), *Update* (Update()) e *Delete* (Remove()). A interface também prevê a implementação de um método Dispose(), para liberação de variáveis da memória.

4.4 IMPLEMENTAÇÃO DO PADRÃO *REPOSITORY* – ACESSO AOS DADOS DE TENANTS

O padrão *Repository* foi implementado para o acesso aos dados de Tenants. Isso envolve dois elementos: a interface ITenantRepository, no projeto Domain.Interfaces, e a classe TenantRepository, em Data.Tenants.

4.4.1 Criação da interface ITenantRepository

A interface ITenantRepository foi apenas uma ponte entre a interface genérica IRepositoryBase<T> e o repositório de dados propriamente dito. A presença dessa interface se faz necessária devido à adição da assinatura do método FindByCredentials(), que pode ser visto (com o restante da interface) no **Anexo E**. Esse método não está previsto em IRepositoryBase<T> e serve para buscar Tenants a partir de suas credenciais – leia-se, usuário e senha.

4.4.2 Criação do repositório de Tenants - TenantRepository

O repositório de Tenants propriamente dito, TenantRepository, está mostrado no **Anexo F**. Nele, foi implementada a interface ITenantRepository. Os métodos desse repositório utilizam o TenantDbContext para acessar a base de dados e operar em cima das tabelas da mesma. O trecho

```
context.Set<Tenant>().Add(obj);
```

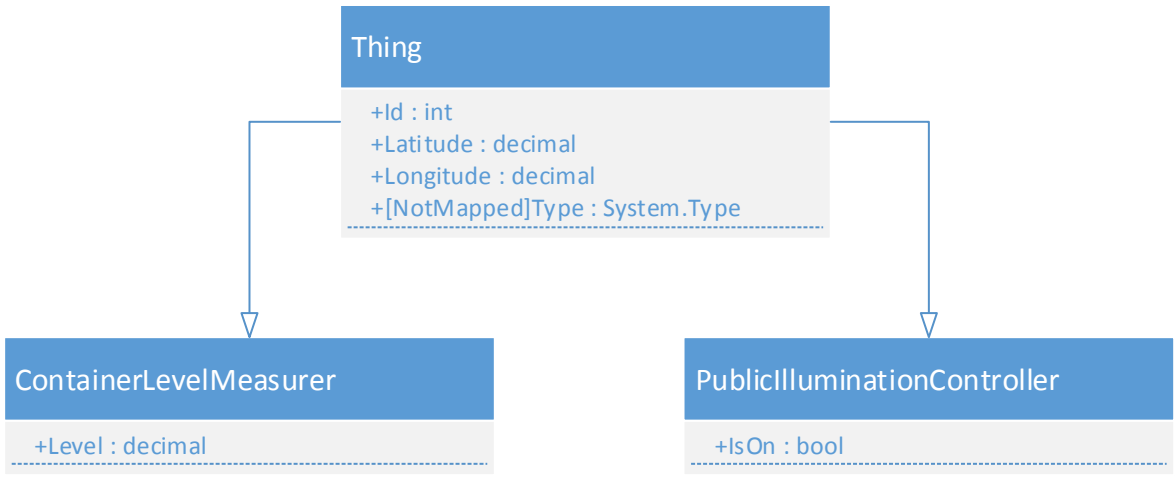
, por exemplo, acessa, a partir do TenantDbContext (variável context), o conjunto de Tenants da base de dados – leia-se "tabela" – adicionado o dado "obj" à essa tabela. Esse trecho é referente ao método Add() do repositório. Os demais métodos seguem a mesma estrutura, encapsulando o acesso ao TenantDbContext e escondendo o mesmo das classes do cliente.

É interessante ressaltar que a grande vantagem do padrão *Repository* está expressa nessa classe: é possível alterar a estrutura do banco de dados sem alterar o código do cliente. Em outras palavras, caso necessário, é possível trocar o acesso aos dados a partir do TenantDbContext, por outra forma de acesso qualquer. Isso facilita a alteração caso não se utilize mais o Entity Framework, por exemplo.

4.5 CRIAÇÃO DA BASE DE DADOS DA APLICAÇÃO

A base de dados da aplicação é focada em *Things*. Em termos de implementação, isso significa que há uma classe base, Thing, que é implementada por cada tipo de "coisa" que está presente na aplicação e será guardada na mesma. Dois tipos estão representados: ContainerLevelMeasurer, uma representação de um medidor do nível de containers de lixo; e PublicIlluminationController, uma representação de um controlador da iluminação pública, indicando se o mesmo está ou não ligado em determinado momento. A **Figura 4.3** mostra em detalhes do relacionamento de herança entre as classes, presentes no projeto Domain.Models.

Figura 4.3 – Diagrama de classes com a herança de Thing



Fonte: Próprio autor

Ambos os tipos se referem a equipamentos que podem ser utilizados em cidades e, como tal, possuem uma localização geográfica baseada em valores de latitude e longitude, além de um identificador único. Note que a aplicação poderia ser facilmente expandida com a adição de mais "coisas", que poderiam representar qualquer equipamento eletrônico presente em uma cidade, desde semáforos a controladores de velocidade, por exemplo.

4.5.1 Criação da classe base Thing (POCO)

Para representação das "coisas" na base de dados, será utilizada uma classe base, Thing, da qual todos os tipos presentes na aplicação derivarão. Essa classe, presente no **Anexo G**, possui as informações comuns a todos os equipamentos que podem ser mapeados pela aplicação: Id, Latitude e Longitude. Além disso, a classe também traz uma quarta propriedade, Type, do tipo System.Type, que não pode ser mapeada pelo Entity Framework (como mostra a *annotation* NotMapped utilizada).

A classe Thing é abstrata, o que indica para o Entity Framework que ela não deve ser construída. Logo, somente suas classes filhas, concretas, representarão entidades do banco de dados.

4.5.2 Criação das classes de "coisas" (POCO) – ContainerLevelMeasurer e PublicIlluminationController

A aplicação que servirá para teste da arquitetura foi criada com duas entidades representando "coisas": um medidor do nível de lixo nos containers e um indicador de ligado/desligado para iluminação pública.

A entidade/classe ContainerLevelMeasurer representa a primeira delas. Essa classe é extremamente simples: além de herdar da classe abstrata Thing (como o fazem todas as "coisas"), contém uma propriedade para identificar o nível de lixo dentro do container. Ela está disponível no **Anexo H**.

Já a entidade/classe PublicIlluminationController, além do que herda da classe abstrata Thing, traz outra propriedade, IsOn, booleano que indica se a luz está acesa ou não. Essa informação é de suma valia para identificar lâmpadas queimadas, por exemplo. A classe está disponível no **Anexo I**.

4.5.3 Criação da SQL Database no Azure

A base de dados foi criada da mesma forma que para os Tenants. Contudo, não foi feita a *migration* para criação da base de dados inicial. Isso ocorre porque os *schemas* e tabelas da base de dados serão criados somente após a subscrição de cada Tenant, separando os dados de cada um.

A string de conexão foi obtida da mesma forma e salva como "IoTDatabase" no arquivo App.config.

4.6 CRIAÇÃO DO *SCHEMA* DE BANCO DE DADOS DO TENANT

A arquitetura de dados escolhida no projeto foi, conforme discutido, *Shared Database, Separated Schemas*. Essa arquitetura permite a separação dos dados de cada Tenant criando-se um *schema* para cada um, com a (s) tabela (s) específicas.

O Entity Framework, em sua Fluent API, trabalha com arquivos de configuração. Esses arquivos de configuração foram criados para permitir a escolha de um *schema* no momento da criação de cada DbSet. Em outras palavras, isso permite que seja escolhido um *schema* diferente para cada tabela da base de dados, caso haja esse interesse.

4.6.1 Criação da configuração para a entidade ContainerLevelMeasurer

O arquivo de configuração traz detalhes do mapeamento de cada entidade/classe para sua respectiva tabela na base de dados. No **Anexo J** observa-se essa configuração.

Primeiramente: herda-se da classe genérica EntityTypeConfiguration, pois ela traz métodos e propriedades que precisamos utilizar para garantir o mapeamento com a Fluent API. Então, utilizando o construtor dessa classe, passamos o *schema* que desejamos para a criação da tabela, com o trecho

```
ToTable("ContainerLevelMeasurer", schema);
```

, que indica que a classe de configuração se refere à criação de uma tabela "ContainerLevelMeasurer" dentro do *schema* que foi passado por parâmetro.

A classe de configuração traz, ainda, outras informações relevantes, como a definição do Id como chave primária e a definição da precisão de Latitude e Longitude (10 bytes, 6 deles após a vírgula) e Level (12 bytes, 1 após a vírgula).

4.6.2 Criação da configuração para a entidade PublicIlluminationController

A configuração do mapeamento entre a classe PublicIlluminationController e a tabela da mesma seguiu a mesma ideia vista anteriormente, com os mesmos valores para chave primária e precisão das propriedades comuns. Nesse caso, não se faz necessário o mapeamento específico da propriedade IsOn, pois ela não precisa de nenhuma configuração extra. O código pode ser visto no **Anexo K**.

4.6.3 Criação do IoTDataContext

O IoTDataContext é muito mais complexo do que o seu equivalente da base de dados de Tenants. Como, para a criação de um *schema* por Tenant, precisa-se dos arquivos de configuração, o DbContext precisa utilizá-los. O código do mesmo pode ser visto no **Anexo L**. Essa classe está presente no namespace Data.IoT.Context.

A classe herda de DbContext, que faz a ponte com a base de dados por nós, lidando com coisas como a conexão e operações na mesma. Para a criação do IoTDataContext, foi utilizado um construtor que recebe uma conexão (DbConnection) e um modelo compilado da base de dados (DbCompiledModel). Esse modelo é utilizado para construir a base de dados em cima do *schema* de cada Tenant.

Outros dois fatores merecem destaque, além dos métodos que fazem a criação do *schema* de cada Tenant: o dicionário "modelCache" é utilizado para guardar, em cache, o modelo para acesso posterior, acelerando o processo; e a propriedade "DbSet<Thing>", que representa a tabela de "coisa" na base de dados. Essa última será transformada na tabela específica para cada "coisa" presente na aplicação – ContainerLevelMeasurer e PublicIlluminationController.

4.6.3.1 Métodos Create() e ProvisionTenant()

Os métodos Create() e ProvisionTenant() são os mais importantes de IoTDataContext. O primeiro deles cria uma instância de IoTDataContext em cima do *schema* do Tenant e do tipo de "coisa" que está sendo compilado. Pode-se notar que o código desse método faz uso do método "modelCache.GetOrAdd()", que retorna o modelo compilado da base de dados para acesso. Esse método utiliza as configurações criadas para ContainerLevelMeasurer e PublicIlluminationController para transformar a propriedade DbSet<Thing> em um DbSet<ContainerLevelMeasurer> ou DbSet<PublicIlluminationController>.

Já o método ProvisionTenant() faz a criação do *schema* para o Tenant, e será chamado somente uma vez, no momento da subscrição do mesmo. Ele utiliza o IoTDataContext criado pelo método Create() para executar um script de criação deste na base de dados.

4.7 IMPLEMENTAÇÃO DO PADRÃO *REPOSITORY* – ACESSO AOS DADOS DA APLICAÇÃO

A implementação do padrão *repository* também se fez necessária para acesso aos dados da aplicação. Esse padrão irá encapsular os dados para serem acessados pelas camadas posteriores, sem que elas saibam o que está acontecendo por baixo, ou seja, escondendo a implementação do banco de dados das camadas superiores.

4.7.1 Criação da interface IThingRepository<T>

Essa interface é apenas uma ponte entre IRepositoryBase<T> e os repositórios concretos de cada "coisa". Ela não adiciona nenhuma informação diferente, mas está presente para adicionar uma camada de abstração para, se necessário no futuro, adicionar-se outros métodos referentes apenas às "coisas" da aplicação. O código pode ser visto no **Anexo M**.

4.7.2 Implementação do repositório ContainerLevelMeasurerRepository

O repositório de ContainerLevelMeasurer's (ContainerLevelMeasurerRepository) pode ser visto no **Anexo N**. Pode-se notar a utilização de um atributo privado do tipo IoTDataContext, que faz o acesso aos dados a partir do repositório. A classe implementa o repositório IThingRepository<ContainerLevelMeasurer>.

O repositório possui dois métodos que merecem destaque: Create() e Get(). O primeiro é responsável pela criação da base de dados para cada Tenant (chamando, simultaneamente, os método Create() e ProvisionTenant() de IoTDataContext). Este será utilizado no momento da subscrição do Tenant. Já o Get() chama apenas o método Create() de IoTDataContext, retornando o repositório para Tenants que já tem sua subscrição feita.

Os demais métodos da classe foram desenvolvidos de forma trivial: chamando o IoTDataContext para adicionar, remover, atualizar e encontrar dados na base de cada Tenant.

4.7.3 Implementação do repositório PublicIlluminationControllerRepository

O repositório de PublicIlluminationController's (PublicIlluminationControllerRepository) foi implementado de forma similar, com os mesmo métodos Get() e Create(). O mesmo pode ser observado no **Anexo O**.

4.8 IMPLEMENTAÇÃO DA SUBCAMADA DE SERVIÇOS

A subcamada de serviços está presente na camada de negócios, ou domínio, da aplicação. Esta serve para abstrair o acesso a vários elementos que são utilizados nas aplicações. A ideia é que, por ser uma aplicação com Arquitetura Orientada a Serviços (SOA), tenha-se esses serviços acessando as informações pertinentes.

A grande vantagem da utilização desse conceito está na separação entre as funcionalidades presentes em cada camada: para a camada de aplicação não interessa o que está acontecendo abaixo do serviço – ela tem apenas essa informação. Trata-se de uma boa prática de programação que faz com que o código tenha uma manutenção e adição de funcionalidades feita de forma muito mais simples.

A subcamada de serviços foi implementada utilizando dois namespaces: Domain.Interfaces.Services, que contém todas as interfaces a ser implementadas para utilização dos serviços; e Domain.Services, que contém as implementações dessas interfaces.

4.8.1 Criação da interface base para os serviços (IServiceBase<T>)

A interface IServiceBase<T> fornece a base genérica a qualquer serviço que será criado, baseando-se em qualquer classe. Nesse caso, os serviços que serão previstos, expostos no **Anexo P**, estão relacionados ao acesso a dados, sendo os seguintes: GetAll(), Add(), Find(), Update() e Remove(), além do sempre necessário Dispose(), que serve, uma vez implementado, para liberar da memória variáveis que não são controladas automaticamente.

4.8.2 Criação da interface base para os serviços de "coisas" (IThingService<T>)

A exemplo do que aconteceu com a interface IThingRepository<T>, essa interface de serviços funciona apenas como uma ponte entre IServiceBase<T> e as implementações de ContainerLevelMeasurer e PublicIlluminationController. Entretanto, está presente para facilitar manutenção e adição de funcionalidades. A mesma está mostrada no **Anexo Q**, que mostra que essa interface apenas herda de IServiceBase<T>.

4.8.3 Criação das interfaces específicas para os serviços (IContainerLevelMeasurerService e IPublicIlluminationControllerService)

As interfaces específicas para os serviços de ContainerLevelMeasurer e PublicIlluminationController são definidas de forma similar ao IThingRepository<T> e, de fato, apenas herdam desta. Estão presentes apenas por questões de manutenção e adição de funcionalidades. O código delas está presente nos **Anexos R** e **S**, respectivamente.

4.8.4 Implementação do serviço base (ServiceBase<T>)

O serviço base, ServiceBase<T>, implementa a interface IServiceBase<T>. Como é possível observar-se, trata-se de uma classe genérica em T. Assim, faz-se uso de um atributo privado e somente leitura "_repository", do tipo IRepositoryBase<T>, para implementar o acesso ao repositório de cada tipo. Os métodos são implementados apenas para fazer a chamada aos métodos do repositório em questão. O código pode ser analisado no **Anexo T**.

4.8.5 Implementação do serviço de "coisas" (ThingService<T>)

O serviço padrão para todas as "coisas" no projeto foi criado de forma mais simples que o anterior, uma vez que herda todas as funcionalidades de ServiceBase<T>. Ainda assim, é preciso sobrescrever o atributo "_repository", bem como o construtor da classe. Agora, estamos utilizando um "IThingRepository<T>", o que condiz com a abstração que está sendo utilizada. O código de ThingService<T> pode ser visto no **Anexo U**.

4.8.6 Implementação dos serviços específicos (ContainerLevelMeasurerService e PublicIlluminationControllerService)

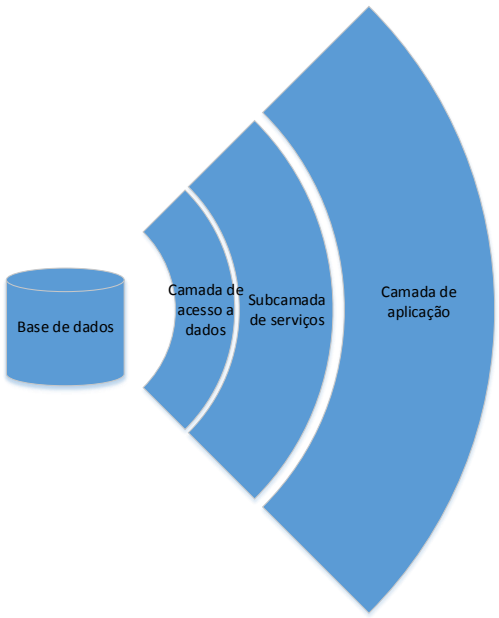
A implementação dos serviços específicos é feita em cima das interfaces IContainerLevelMeasurerService e IPublicIlluminationControllerService. Além dessa implementação, as classes herdam de ThingService<T>, onde T, nesse caso, é o respectivo tipo de cada uma: ContainerLevelMeasurer e PublicIlluminationController. A ideia é que, através do uso de *generics*, possa-se fazer uso dos métodos implementados na classe ServiceBase<T>.

Além disso, também foi necessário alterar a implementação do construtor da classe, a exemplo do que foi feito anteriormente. Entretanto, como mostram os códigos nos **Anexos V** e **W**, o parâmetro T não é mais genérico, e tem-se um IThingRepository<ContainerLevelMeasurer> em ContainerLevelMeasurerService e um IThingRepository<PublicIlluminationController> em PublicIlluminationControllerService.

4.9 IMPLEMENTAÇÃO DA CAMADA DE APLICAÇÃO

A camada de aplicação fornece a abstração para todo acesso a funcionalidades de qualquer tipo que serão feitas pelas aplicações. Nesse caso, utilizamos essa camada para abstrair o acesso aos dados de "coisas", seguindo a estrutura mostrada na **Figura 4.4**. Nota-se que a mesma faz a hierarquia entre as camadas, onde a Camada de Aplicação pede informações ao domínio, através da Subcamada de Serviços, e esta última conversa com o repositório para obter os dados pedidos.

Figura 4.4 – Acesso aos dados através das camadas



Fonte: Próprio autor

4.9.1 Criação da interface base para os serviços da aplicação (IAppServiceBase<T>)

Os serviços da aplicação possuem uma base muito similar ao visto nos serviços da subcamada de serviços. A diferença é que eles operam em uma camada acima e terão contato direto com nossa aplicação, ou seja, a camada de apresentação.

Como tal, a interface genérica IAppServiceBase<T> prevê os mesmos métodos de acesso a dados: GetAll(), Find(), Add(), Update() e Remove(). Isso está mostrado no **Anexo X**.

4.9.2 Criação da interface base para os serviços da aplicação de "coisas" (IThingAppService<T>)

Assim como sua “irmã” da subcamada de serviços, a interface IThingAppService<T> é genérica e não possui nenhuma assinatura de método adicional. Ela está presente apenas para questões de manutenção e adição de funcionalidades. A mesma herda de IAppServiceBase<T> e está no **Anexo Y**.

4.9.3 Criação das interfaces específicas para os serviços da aplicação (IContainerLevelMeasurerAppService e IPublicIlluminationControllerAppService)

Da mesma forma como a interface genérica IThingAppService<T>, as interfaces da camada de aplicação para ContainerLevelMeasurer e PublicIlluminationController não preveem a implementação de nenhum método extra: apenas herdam de IThingAppService<T> e, consequentemente, de IAppServiceBase<T>. IContainerLevelMeasurerAppService e IPublicIlluminationControllerAppService estão presentes nos **Anexos Z** e **AA**, respectivamente.

4.9.4 Implementação do serviço base da aplicação (AppServiceBase<T>)

A classe AppServiceBase<T> é uma implementação da interface IAppServiceBase<T>, basicamente. Esta faz o acesso a um serviço, através de um atributo protegido do tipo IServiceBase<T>. Este é atribuído no construtor da classe.

Os métodos implementados em AppServiceBase<T> utilizam esse atributo protegido, “_serviceBase”, para acessar os respectivos métodos na subcamada de serviços e mostrar os dados para os clientes. O código da classe está mostrado no **Anexo AB**.

4.9.5 Implementação do serviço de “coisas” da aplicação (ThingAppService<T>)

A classe ThingAppService<T> foi implementada utilizando o conceito de herança: a classe herda tudo que foi implementado em AppServiceBase<T>. Além disso, essa classe também implementa a interface IThingAppService<T>.

Além dos métodos herdados, a classe prevê um construtor que irá popular um atributo do tipo IThingService<T>, “_service”, que será utilizado para acessar os respectivos métodos na subcamada de serviços. O código pode ser observado no **Anexo AC**.

4.9.6 Implementação dos serviços específicos da aplicação (ContainerLevelMeasurerAppService e PublicIlluminationControllerAppService)

Essas classes possuem implementações mais complexas que as anteriores, como mostram os códigos nos **Anexos AD** e **AE**, respectivamente. Ambas possuem um método Factory() que, utilizando o *schema* do Tenant e uma DbConnection, busca o repositório de dados e, posteriormente, o serviço, retornando uma instância do serviço da aplicação que será utilizado para acesso aos dados.

4.10 ADIÇÃO DE CRIPTOGRAFIA PARA SALVAMENTO DE SENHAS

A criptografia é parte essencial em qualquer sistema que lida com senhas. Por padrão, o sistema de autenticação e autorização utilizado no projeto utiliza uma *hash string*, base 64. Contudo, por ser base 64, tal criptografia é de fácil quebra, caso alguma pessoa indesejada ponha as mãos na senha criptografada. Pensando nisso, foi feita a substituição dessa política por outra, mais eficiente, baseada no algoritmo MD5.

4.10.1 Criação da criptografia baseada em MD5

A criptografia, no OWIN, é baseada em um "Password Hasher", ou seja, uma entidade capaz de transformar uma string comum em uma *hashed string*, baseada em algum algoritmo. Para alterar essa política, bastou a criação de uma classe, SimplePasswordHasher (disponível no **Anexo AF**), implementando a interface IPasswordHasher. Essa classe foi criada no namespace Common.Infrastructure.Cryptography.

É possível notar-se a implementação de dois métodos: HashPassword() e VerifyHashedPassword(). O primeiro retorna a string criptografada, após receber a versão *plain text* da mesma. Já o segundo faz a comparação entre os valores, indicando se as senhas se equivalem.

4.11 PREPARAÇÃO DO SITE PARA SUBSCRIÇÃO DOS TENANTS

A arquitetura Multi-Tenant se baseia em uma situação muito simples: os Tenants precisam fazer a subscrição para utilização do serviço. Isso faz com que a arquitetura seja muito utilizada em aplicações para SaaS, e no caso do projeto criado não é diferente. Os Tenants precisam fazer a subscrição e, uma vez que isso está concluído, é realizada a criação da base de dados para cada um deles.

O primeiro passo foi a criação da política de segurança dos Tenants, utilizando o Microsoft.Owin. Essa biblioteca tira a complexidade na utilização dos conceitos de autorização e autenticação, delegando as tarefas mais complexas a esse respeito ao framework. Para a subscrição dos Tenants, foi utilizado um UserManager, responsável pela criação do Tenant na base de dados e sua "inscrição" na aplicação. Após essa etapa, a base de dados do Tenant pode ser criada, utilizando o seu Id como *schema*.

Como é possível observar-se, esse website será muito simples, possuindo apenas uma tela com um formulário de inscrição de Tenant. Nessa tela, o Tenant irá informar que tipo de "coisa" ele deseja: um ContainerLevelMeasurer ou um PublicIlluminationController. Essa escolha irá definir a base de dados que será criada para ele e, consequentemente, os elementos que estarão disponíveis na aplicação para a acesso e amostragem.

O site será criado utilizando o ASP.NET MVC.

4.11.1 Criação do SubscriptionTenantManager (UserManager)

O Microsoft.Owin faz uso de um UserManager<T> para definição dos detalhes referentes aos usuários de cada aplicação que serão autenticados e, posteriormente, autorizados ou não. Para adição dos

detalhes necessários na aplicação Multi-Tenant, foi feita a criação de um SubscriptionTenantManager, disponível no **Anexo AG**. Nesse código, tem-se algumas definições:

- Construtor que recebe um IUserStore<Tenant>: como o nome sugere, o construtor recebe um armazenador de Tenants, que será utilizado como o repositório de dados;
- Método Create(), que retorna um SubscriptionTenantManager com algumas definições. As definições nesse método são importantes, pois foi definido que o repositório de Tenants é um TenantRepository e, além disso, que o método para criptografia das senhas é o que foi criado – SimplePasswordHasher.

4.11.2 Inscrição do SubscriptionTenantManager e seu repositório de dados

O recém-criado SubscriptionTenantManager precisa ser inscrito para que o Owin saiba que se trata do UserManager padrão do projeto. Essa configuração é feita na classe Startup do projeto, que está presente no **Anexo AH**, através do método CreatePerOwinContext() de IAppBuilder.

4.11.3 Criação do TenantViewModel

As aplicações em geral trabalham com um conceito muito importante e interessante: as ViewModels. Esses modelos são especialmente úteis no ASP.NET MVC, quando os dados precisam ser passados entre a view e o controller através de POST's, PUT's e afins. Nesse caso, foi utilizado um HTTP POST para a passagem dos dados para criação do Tenant entre cliente e servidor. Assim, é preciso que haja um ViewModel para armazenar esses dados e realizar o *binding* entre eles de forma correta.

É aí que entre o TenantViewModel. Essa classe, mostrada no **Anexo AI**, prevê 5 propriedades: Username, Email, Password, ConfirmPassword e Type. Os primeiros 4 fazem exatamente o que o nome sugere, auxiliados por uma série de DataAnnotations que garantem que os dados sejam validados ainda no cliente. A última representa uma enumeração chamada ThingType, que contém os valores para ContainerLevelMeasurer e PublicIlluminationController, que irão indicar que tipo de Tenant está sendo criado e com que tipo de dados ele irá lidar após a subscrição.

4.11.4 Criação da view para criação de Tenants

O ASP.NET MVC ou, mais especificamente, o *view engine* Razor, possui um conceito chamado de views fortemente tipadas. Essas views aceitam um **model**, ou seja, um modelo base sobre o qual elas irão trabalhar e que permite que dados sejam recebidos e/ou passados entre a view e o controller. Nesse caso, a view CreateUser.cshtml é fortemente tipada, e seu tipo é um TenantViewModel.

Com base nesse tipo, tem-se a definição de um formulário, onde utilizam-se Html Helpers para criação dos labels e campos de inserção de dados automaticamente. O código está mostrado no **Anexo AJ**, e a view criada pode ser vista na **Figura 4.5**.

Figura 4.5 – Tela de subscrição de Tenants

Fonte: Próprio autor

O ponto que merece destaque nessa view é o helper Html.BeginForm(), que cria o formulário e faz o POST dos dados para o servidor no momento do submit. É assim que o controller recebe esses dados e os trata para armazenar o Tenant e criar sua base de dados.

4.11.5 Criação do controller para criação de Tenants (AccountController)

O AccountController, disponível no **Anexo AK**, como sugere o MVC Framework, é responsável por lidar com todas as requisições transmitidas pela view. No caso do POST do formulário de criação de usuários, não é diferente. Entretanto, alguns pontos precisam ser destacados a respeito desse controller:

1. AccountController possui 2 campos privados: _clmRepository (o repositório de ContainerLevelMeasurer) e _picRepository (o repositório de PublicIlluminationController), sendo que somente um deles será criado, dependendo da escolha do Tenant;
2. O controller também possui uma propriedade TenantManager, que retorna o gerenciador SubscriptionTenantManager criado anteriormente;

Esse controller ainda possui dois Action Methods, ambos CreateUser(): um deles é o método GET, que simplesmente retorna a view; o outro é o POST, que recebe os dados do formulário e trata-os. Esse último tem um parâmetro do tipo TenantViewModel, que retorna os dados do formulário e, caso sejam válidos, faz as operações em ordem:

1. Cria um Tenant com os dados passados;
2. Utiliza o TenantManager para criar o Tenant na base de dados;
3. Caso a criação seja bem-sucedida, chama o repositório escolhido e cria a base de dados, utilizando o Id do recém-criado Tenant como *schema*;
4. Ao final, redireciona para a aplicação.

O controller ainda possui um método auxiliar, AddErrors(), que serve para adicionar os erros ao modelo e mostra-los para o usuário.

4.12 PREPARAÇÃO DO SITE PARA A APLICAÇÃO – FRONT-END

Essa aplicação, definida no website Site.Application, é o cliente web do projeto. Esse cliente web visa mostrar os dados de cada Tenant de uma forma específica, de acordo com o tipo escolhido por ele. Esse tipo de amostragem é uma das grandes vantagens da arquitetura Multi-Tenant, que permite que cada Tenant possua uma certa customização nas views que lhe são apresentadas. Nesse caso, a customização está no tipo de dados que será amostrado.

Em um primeiro momento, foram preparados os elementos para login e amostragem de dados de cada Tenant. Para isso, primeiramente, foi implementada a política de segurança na aplicação, ainda com o auxílio do OWIN. A base de dados de cada Tenant foi buscada através dos AppServices criados. Em outras palavras, a hierarquia de camadas foi obedecida, pois a Camada de Apresentação só conhece a Camada de Aplicação, e faz toda a comunicação com os dados através dela.

O website inteiro está baseado em um layout que utiliza o Bootstrap. O layout é extremamente simples, mas serve ao propósito de organizar os elementos e deixa-lo visualmente mais interessante para os usuários da aplicação. Em uma versão para *release* dessa aplicação, contudo, não é interessante utilizar esse visual, devido a ser pouco atrativo para os clientes.

4.12.1 Implementação da política de segurança na aplicação

A política de segurança no acesso é dos pontos mais importantes em qualquer aplicação Multi-Tenant. É preciso garantir, com 100% de certeza, que um Tenant não terá acesso a nada do que é mostrado a outro Tenant. Isso envolve as personalizações, dados e outros elementos quaisquer que possam estar envolvidos com cada Tenant.

Para implementação dessa política, foi utilizado o OWIN, uma vez mais. Essa biblioteca possui, além do UserManager já comentado, o conceito do SignInManager: uma entidade sobre a qual delega-se a tarefa de fazer o SignIn/LogIn dos Tenants na aplicação. Para essa implementação estar completa, é preciso registrar ambos os elementos na aplicação, na classe Startup.

4.12.1.1 Criação do ApplicationTenantManager

O ApplicationTenantManager não possui nenhuma mudança relevante com relação ao SubscriptionTenantManager. Seu código está no **Anexo AL**.

4.12.1.2 Criação do ApplicationSignInManager

O ApplicationSignInManager herda de uma classe base do OWIN: a classe genérica SignInManager<Tenant, string>. O primeiro atributo genérico, é claro, é o Tenant que será logado; o segundo representa a chave de login/signin, ou seja, a senha. O código do **Anexo AM** mostra em detalhes como funciona o ApplicationSignInManager:

- A estrutura é similar ao visto no ApplicationTenantManager: um construtor e um método Create();
- O construtor recebe o ApplicationTenantManager e uma implementação da interface IAuthenticationManager – que contém as informações de autenticação para SignIn/LogIn.
- Já o método Create() recebe as opções de SignIn e o IOwinContext, retornando um novo ApplicationSignInManager com essas informações.

4.12.1.3 Registro dos gerenciadores de SignIn e Tenant

O registro dos gerenciadores de SignIn e Tenant foi feito, a exemplo do projeto Site.Subscription, na classe Startup (**Anexo AN**). O método Configuration() dessa classe chama um outro método, ConfigureAuth(), presente na classe Startup.Auth.

4.12.1.3.1 Criação do Startup.Auth

O método ConfigureAuth da classe Startup.Auth é o único presente na classe. Trata-se de uma extensão da classe parcial Startup, e por isso o método está acessível diretamente da classe Startup, sem a necessidade de utilização de objetos ou métodos estáticos. É uma extensão da mesma classe desenvolvida em um novo arquivo.

Sobre o método em si, foi utilizado o atributo IAppBuilder para chamar o método CreatePerOwinContext(), passando, respectivamente, o TenantDbContext (através do método Create() do TenantRepository), o ApplicationTenantManager e o ApplicationSignInManager. Por fim, foi definida a forma de autenticação: utilizando cookies. Isso foi realizado através do método UseCookieAuthentication(), passando uma série de configurações. Essas configurações indicam o caminho do Login, para o qual será redirecionado todo Tenant não autorizado, o nome do cookie, tempo de expiração do mesmo, entre outras informações relevantes que podem ser observadas no **Anexo AO**.

4.12.2 Implementação do login

O Login é, talvez, a parte mais essencial da segurança da aplicação. É aqui que estaremos garantindo que cada Tenant tenha acesso somente às suas views e seus dados, sem invadir o “espaço” dos demais. Logo, essa implementação precisa ser totalmente à prova de falhas. E é aí que entra o ApplicationSignInManager: foram utilizados os métodos dessa classe (herdados de SignInManaget<Tenant, string>) para realização de LogIn/LogOff.

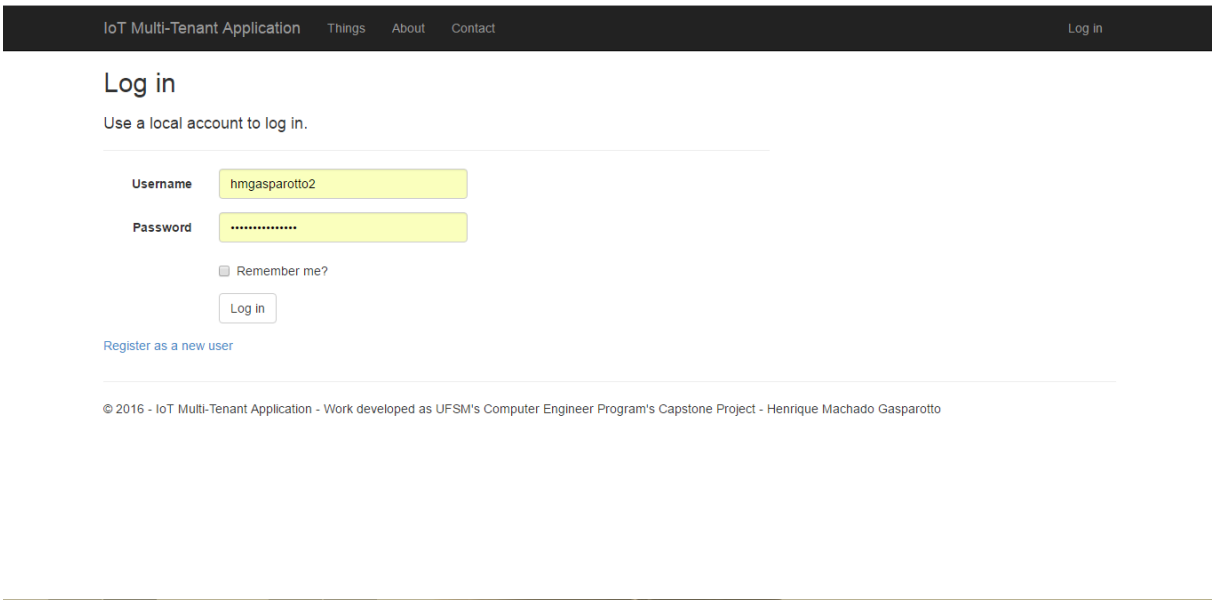
4.12.2.1 Criação do LoginViewModel

O primeiro passo para a criação da estrutura de Login é o LoginViewModel. Essa classe é a representação de tudo que estará presente no formulário de Login, permitindo que os dados enviados via HTTP POST sejam entendidos pelo AccountController. Podemos observar o código dessa classe no **Anexo AP**. Nota-se que o mesmo possui várias DataAnnotations para a criação automática da view utilizando os Html Helpers do MVC.

4.12.2.2 Criação da tela de login

A view Login.cshtml é relativamente simples: utilizando o helper Html.BeginForm(), faz-se o login com as informações do LoginViewModel. Essa view é fortemente tipada e seu tipo é LoginViewModel. Um

Figura 4.6 – View de login



ponto que merece destaque é o link para registro de um novo Tenant, que faz a ligação entre a Application e a Subscription. O código pode ser visto no **Anexo AQ**. A tela pode ser vista na **Figura 4.6**.

Fonte: Próprio autor

4.12.2.3 Criação do AccountController

O AccountController, disponível no **Anexo AR**, é o elemento central (como convém ao MVC) que faz o login propriamente dito. Para isso, tem-se dois campos privados: "_signInManager" e "_tenantManager", dos tipos ApplicationSignInManager e ApplicationTenantManager, respectivamente. Esses campos geram duas propriedades que serão acessadas dentro do controller para utilizar as classes em questão.

No código, alguns elementos merecem destaque: os métodos GET e POST de Login e o método POST de LogOff.

4.12.2.3.1 Login

O primeiro deles é o método GET de Login(), que recebe uma URL de retorno por parâmetro. Com essa URL, o POST de Login() consegue redirecionar o Tenant para a página que ele tentou acessar e não tinha autorização para tanto. Já o HTTP POST de Login() é mais complexo, recebendo um LoginViewModel com os dados do formulário. Basicamente, caso o modelo seja válido (não haja erros de autenticação nos dados do cliente, o SignInManager fará um "PasswordSignInAsync()", que nada mais é do que o SignIn do Tenant utilizando o usuário e a senha do mesmo.

4.12.2.3.2 LogOff

Também há o método de LogOff(), que utiliza o AuthenticationManager para realizar o SignOut(). O AuthenticationManager é uma propriedade que retorna a propriedade Authentication do OwinContext.

4.12.3 Criação do HomeController

O HomeController é criado de forma padronizada pelo template do Microsoft Visual Studio, e foi feita uma alteração nele: a retirada do Action Method Index(). Os demais métodos permanecem idênticos. O código do HomeController pode ser visto no **Anexo AS**. É importante notar-se a Annotation "[Authorize]" – é ela que indica ao Owin que se trata de um controller com métodos acessíveis apenas a Tenants logados.

4.12.3.1 Criação da view About

A view About.cshtml também é criada de forma padronizada pelo template do Visual Studio. Entretanto, ela foi alterada para trazer informações a respeito do projeto desenvolvido. O código pode ser visto no **Anexo AT**. A view pode ser vista na **Figura 4.7**.

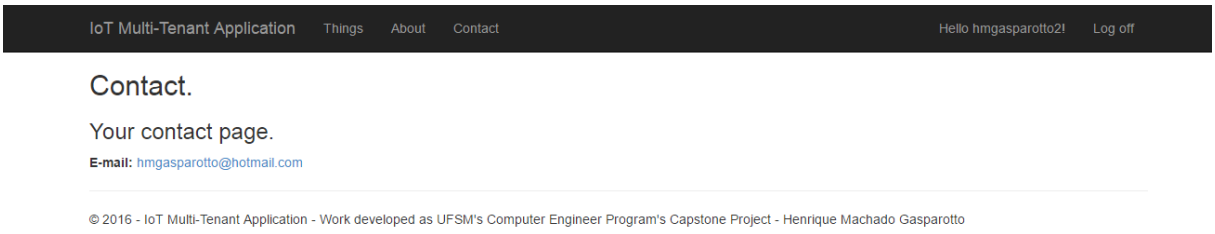
Figura 4.7 – View Home/About

Fonte: próprio autor

4.12.3.2 Criação da view Contact

A view Contact.cshtml também é criada automaticamente e, a exemplo da view About, foi alterada para receber informações relevantes ao projeto. O código pode ser visto no **Anexo AU**. A view pode ser vista na **Figura 4.8**.

Figura 4.8 – View Home/Contact



Fonte: próprio autor

4.12.4 Criação do ThingController

O ThingController é, provavelmente, a classe mais complexa do projeto. E isso é esperado, uma vez que este centraliza todo o acesso dos Tenants aos dados e, para isso, utiliza os dados do Tenant logado e acessa o AppService de cada um deles. Com isso, o mesmo precisa de 4 atributos privados: os serviços de aplicação para ContainerLevelMeasurer (IContainerLevelMeasurerAppService) e PublicIlluminationController (IPublicIlluminationControllerAppService), além do ApplicationTenantManager e do Tenant logado.

Além disso, como pode ser visto no código do **Anexo AV**, o controller traz métodos para criação, edição, detalhes e remoção das "coisas" da base de dados do Tenant. Ou seja, o mesmo tem controle total sobre as suas Thing's através desse controle e do acesso que este tem ao AppService específico de cada um. O controller traz ainda um método, RepositoryGetter(), responsável por inicializar os atributos privados para serem utilizados dentro de cada método.

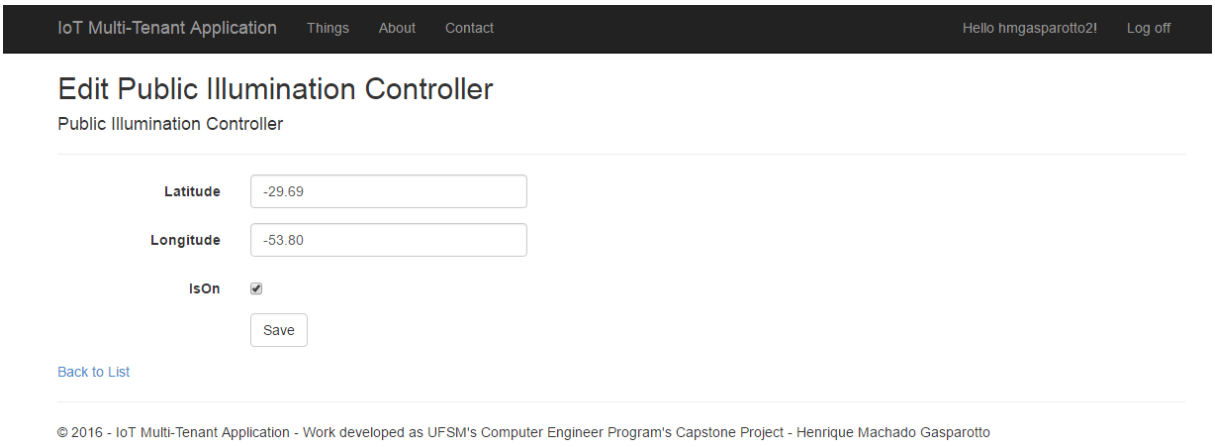
4.12.4.1 Acesso aos dados de cada Tenant

O acesso aos dados de cada Tenant é feito no método RepositoryGetter(), que busca, utilizando o tipo de Tenant, o AppService específico para cada um. Para isso, é utilizado o método Factory() de cada AppService. Uma vez que o atributo _clmService/_picService exista, todos os métodos deles estão disponíveis, fazendo com que o Tenant possa realizar qualquer alteração/consulta ao serviço.

4.12.4.2 Criação das views de edição

As views de edição (EditContainerLevelMeasurer.cshtml e EditPublicIlluminationController.cshtml) são praticamente idênticas. A diferença está em seu tipo: ambas são fortemente tipadas e possuem tipos ContainerLevelMeasurer e PublicIlluminationController, respectivamente, como seus nomes sugerem. Elas utilizam o helper Html.BeginForm() para passagem dos

Figura 4.9 – View Thing/EditPublicIlluminationController



dados de edição de cada Thing via HTTP POST. Os códigos podem ser observados nos **Anexos AW** e **AX**. A view de edição para PublicIlluminationController's pode ser vista na **Figura 4.9**.

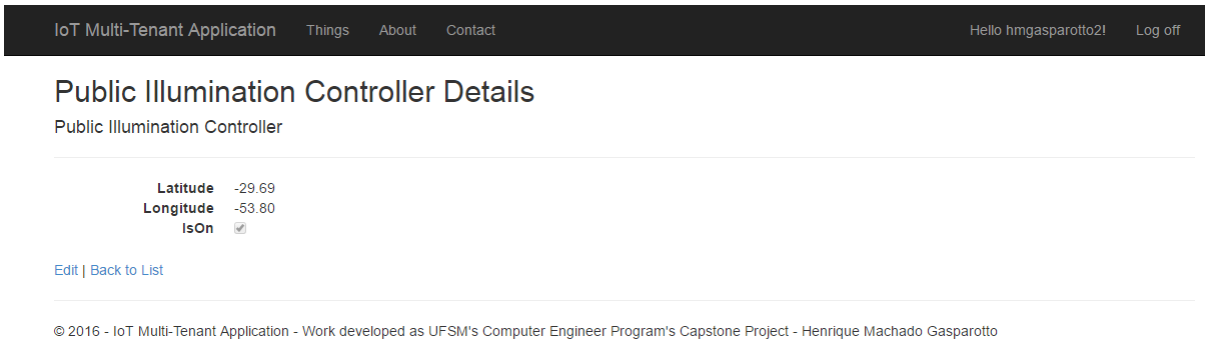
Fonte: próprio autor

4.12.4.3 Criação das views de criação

As views de criação (CreateContainerLevelMeasurer.cshtml e CreatePublicIlluminationController.cshtml) também são praticamente idênticas, sendo diferenciadas pelo seu tipo. A estrutura é similar às views de edição e seus códigos podem ser observados nos **Anexos AY** e **AZ**.

4.12.4.4 Criação das views de detalhes

Figura 4.10 – View Thing/DetailsPublicIlluminationController



Fonte: próprio autor

As views de detalhes (DetailsContainerLevelMeasurer.cshtml e DetailsPublicIlluminationController.cshtml) não possuem nenhum HTTP POST, o que significa que não possuem nenhum formulário a ser preenchido. Assim, sua estrutura é um pouco diferente das views de criação, seu código pode ser visto nos **Anexos BA** e **BB** e sua view pode ser vista na **Figura 4.10**.

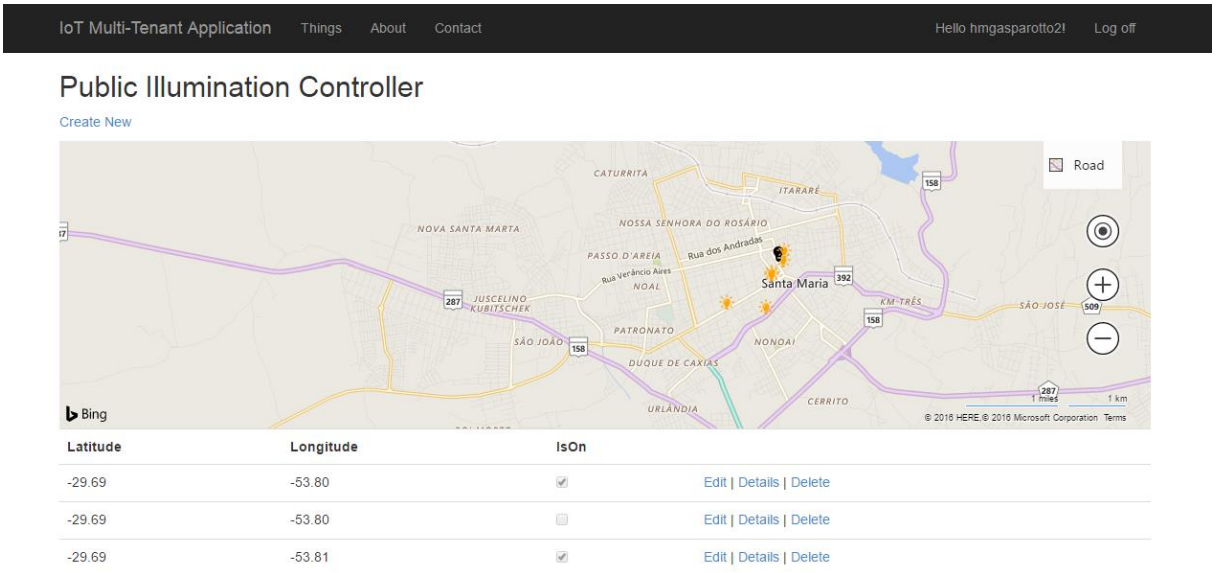
4.12.4.5 Criação das views de remoção

As views de remoção (DeleteContainerLevelMeasurer.cshtml e DeletePublicIlluminationController.cshtml) possuem a mesma estrutura da view de detalhes, como mostram os **Anexos BC** e **BD**.

4.13 ADIÇÃO DO BING MAPS NA APLICAÇÃO

A aplicação que foi criada se baseia em “coisas” que possuem elementos em comum: posição geográfica. Assim, ressalta-se que a visualização da posição desses elementos em um mapa é de suma importância na mesma. É aí que entra o Bing Maps, escolhido por sua simplicidade, robustez e visual atrativo. Essa solução permite a colocação de um mapa em qualquer página web, com visualização customizada utilizando apenas código JavaScript. O mapa foi adicionado na página de Index de cada Thing na aplicação, com pushpins representando os locais dessas entidades através de ícones personalizados e que indicam o estado das mesmas: nível alto ou baixo, no caso dos containeres, e ligado/desligado, no caso dos indicadores de iluminação pública. A **Figura 4.11** mostra o Bing Maps em ação na view.

Figura 4.11 – View Thing/IndexPublicIlluminationController com Bing Maps

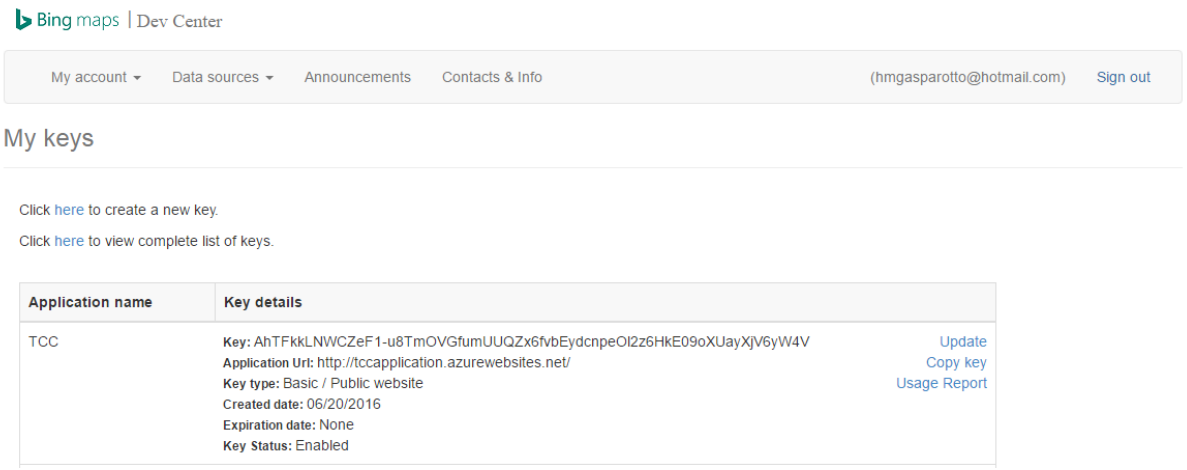


Fonte: próprio autor

4.13.1 Registro no Bing Maps (obtenção de credencial)

O registro no Bing Maps é feito através da URL https://www.bingmapsportal.com/Application, mostrada na **Figura 4.12**. A criação da chave é feita de forma bastante simples, apenas seguindo as instruções mostradas no website. Então, basta utilizar a credencial retornada e adicioná-la na aplicação.

Figura 4.12 – Registro no Bing Maps (obtenção de credenciais)



Fonte: próprio autor

4.13.2 Utilização do mapa nas views Index

De posse da credencial obtida, foi possível inserir um mapa na view Index de cada Thing. Essa view, também fortemente tipada, recebe uma coleção de valores de Thing, mostrando os seus dados tanto no mapa, através dos pushpins, como em uma lista logo abaixo deste. O código das views IndexContainerLevelMeasurer.cshtml e IndexPublicIlluminationController.cshtml podem ser observados nos **Anexos BE** e **BF**.

O trecho JavaScript

```
map = new Microsoft.Maps.Map(document.getElementById('mapViewer'), {
    credentials: 'AhTFkkLNWCZeF1-u8TmOVGfumUUQZx6fvbEydcnpeOl2z6HkE09oXU-
ayXjV6yW4V',
    center: new Microsoft.Maps.Location(-29.691037,-53.835232),
    zoom: 13
});
```

inicializa o mapa, centralizando-o em uma posição central na cidade de Santa Maria, RS – indicada pelos valores de latitude e longitude passados no atributo “center”.

4.13.3 Utilização de pushpins com ícones personalizados

A utilização de ícones personalizados também é executada através de um código JavaScript. O trecho

```
@foreach (var location in Model)
{
    <text>
```

```
                var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Loca-
tion(@location.Latitude, @location.Longitude), { icon: @{ if (location.IsOn)
{ <text>'Content/Images/light.png'</text> } else { <text>'Content/Images/light-
off.png'</text> } } });
                map.entities.push(pushpin);
                Microsoft.Maps.Events.addHandler(pushpin, 'click', function () {
                    window.location.href = "@Url.Action("DetailsPublicIlluminationCon-
troller", new { Id = location.Id })";
                });
                </text>
            }
```

faz a iteração entre todos os elementos da coleção de Thing's passada para a view e cria um pushpin para cada uma delas. É interessante notar-se que o ícone escolhido depende da propriedade IsOn do objeto (referente ao PublicIlluminationController). O trecho final adiciona um *handler* (addHandler()) para o evento do clique no pushpin, fazendo com que o mesmo leve o Tenant para a tela de detalhes da Thing.

4.13.4 Carregamento assíncrono do mapa

O carregamento do mapa é feito de forma assíncrona graças aos atributos HTML5 "async" e "defer", no trecho

```
<script
src="http://www.bing.com/api/maps/mapcontrol?branch=release&callback=loadMapScenario"
async defer></script>
```

. Esse trecho traz o script que realiza o load do map propriamente ditto, recebendo no atributo "callback" o método a ser chamado para carregamento do mesmo.

4.13.5 Criação de "coisa" no clique em posição geográfica escolhida

A criação da Thing é realizada de duas formas: através do link "Create New", no canto superior esquerdo, e do clique em qualquer posição do mapa. Essa segunda opção utiliza AJAX para chamar a *action* Create<Thing>(), como mostra o trecho a seguir:

```
            Microsoft.Maps.Events.addHandler(map, 'click', function (e) {
                var point = new Microsoft.Maps.Point(e.getX(), e.getY());
                var loc = e.target.tryPixelToLocation(point);
                var location = new Microsoft.Maps.Location(loc.latitude, loc.longi-
tude);
                var PublicIlluminationController =
                {
                    "Latitude": location.latitude,
                    "Longitude": location.longitude,
                    "IsOn": true
                };
                $.ajax({
                    url: '/Thing/CreatePublicIlluminationController/',
                    data: JSON.stringify(PublicIlluminationController),
                    type: 'POST',
                    contentType: 'application/json; charset=utf-8'
                });
                window.location.href = "@Url.Action("Index")";
            });
```

O primeiro trecho do código obtem a posição geográfica na qual o clique foi realizado. É importante notar que esse método não retorna os valores de forma 100% precisa, embora se aproxime muito disso. Após, faz a chamada AJAX ao método, passando um model criado com os dados. Por fim, atualizamos a página para que os dados mostrados sejam sempre os mais atualizados possíveis.

4.14 CRIAÇÃO DA WEB API PARA ACESSO EXTERNO

Uma das grandes vantagens da utilização da SOA é a possibilidade de compartilhar os mesmos elementos de serviços entre os muitos clientes. Esse tipo de abordagem é cada vez mais importante, pois uma aplicação não pode pensar em sobreviver em um mundo focado em smartphones sem um cliente Android e/ou iOS, por exemplo. É nesse ponto que entra a Web API criado no projeto. A ideia é que tenhamos a possibilidade de criar um cliente externo que seja capaz de acessar os mesmos dados através desse cliente, protegido por um conceito chamado de Bearer Token.

4.14.1 Implementação de segurança no acesso à API

A segurança no acesso à web API foi criada para ser controlada através de tokens. Esse tipo de autenticação é bastante comum em APIs, devido ao fato de serem criados apenas uma vez e serem, então, utilizados praticamente para todo o sempre. A grande questão é evitar que esses dados caiam em mãos erradas, o que não é tão difícil assim. Um cliente Android, por exemplo, poderia pedir um token na primeira vez que executa a aplicação e salvá-lo em sua base de dados local. Isso garantiria segurança ao token, que só seria utilizado se o usuário perdesse ou tivesse o celular roubado.

Para implementar esse tipo de segurança, foi utilizado o OWIN uma vez mais. Essa biblioteca possui diferentes formas de autenticação, e o Bearer Token é uma delas. Para essa implementação, novamente foi utilizada a classe Startup, agora sem a necessidade de “Managers”, como mostra o código do **Anexo BG**. Nota-se, no método ConfigureOAuth(), a criação de um servidor de tokens, com as opções sendo passadas. Uma dessas opções é a definição do TokenEndpointPath, que é a rota que será utilizada para criação de tokens. Outras duas definições importantes são: AccessTokenExpireTimeSpan, que indica o tempo no qual o token expirará (100 anos, no caso), e Provider, que define o provedor para geração de tokens.

Esse provedor é extremamente importante, pois cuida da geração dos tokens e, logicamente, da autenticação do Tenant que está requisitando esse token. Isso pode ser notado claramente no código do **Anexo BH**. Esse provedor traz dois métodos sobrescritos: ValidateClientAuthentication(), que valida o token informado pelo Tenant, e GrantResourceOwnerCredentials(), que é executado quando a rota “/token” é chamada. Aqui, o provedor valida as credenciais e cria o access token, retornando-o para o usuário na forma de um JSON.

4.14.2 Criando API Controllers – ContainerLevelMeasurerController e PublicIlluminationControllerController

Os controllers da API não são muito diferentes dos da Application. Seu funcionamento é similar, fazendo as operações através de um AppService (PublicIlluminationAppService ou ContainerLevelMeasurerAppService) e identificando o Tenant e questão, validando-o contra o repositório de Tenants.

As rotas criadas também são triviais: o método Get() é mapeado para a rota "api/publicilluminationcontroller" ou "api/containerlevelmeasurer". Outras rotas, como "api/<thing>/{id}", podem ser mapeadas de diversas formas, dependendo do método HTTP utilizado. O código completo dos dois controller pode ser visto nos **Anexos BI** e **BJ**, e é de fácil entendimento, no geral. Entretanto, alguns métodos merecem destaque, ambos na região "Helpers" ao final.

4.14.2.1 Método IdentityGetter()

O método IdentityGetter() é responsável por "quebrar" o access token para identificar o usuário no qual esse token é baseado. Embora a prática de quebrar o access token não seja comum, é necessário para saber a qual Tenant estamos nos referindo e, consequentemente, para trazer o AppService correto para acesso aos dados.

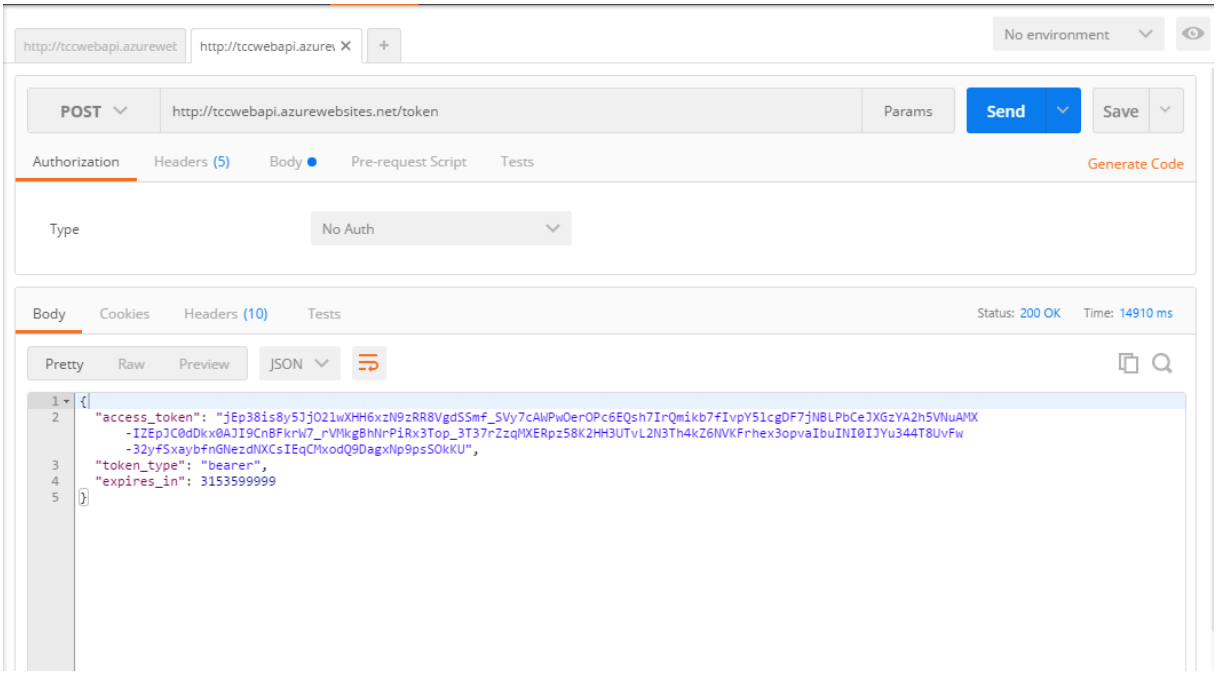
4.14.2.2 Método RepositoryGetter()

O método RepositoryGetter() popula o atributo _picService ou _clmService, utilizando o AppService e seu método Factory().

4.14.3 Teste das rotas criadas

O teste das rotas criada foi realizado utilizando o Postman, ferramenta disponível da Chrome Web Store, do Browser Google Chrome. A **Figura 4.13** mostra o teste da rota para geração de tokens. Essa rota recebe as informações do Tenant e retorna um access token específico para ele. Nota-se, também, que é retorando o tipo de token e o tempo de vida do mesmo, em segundos.

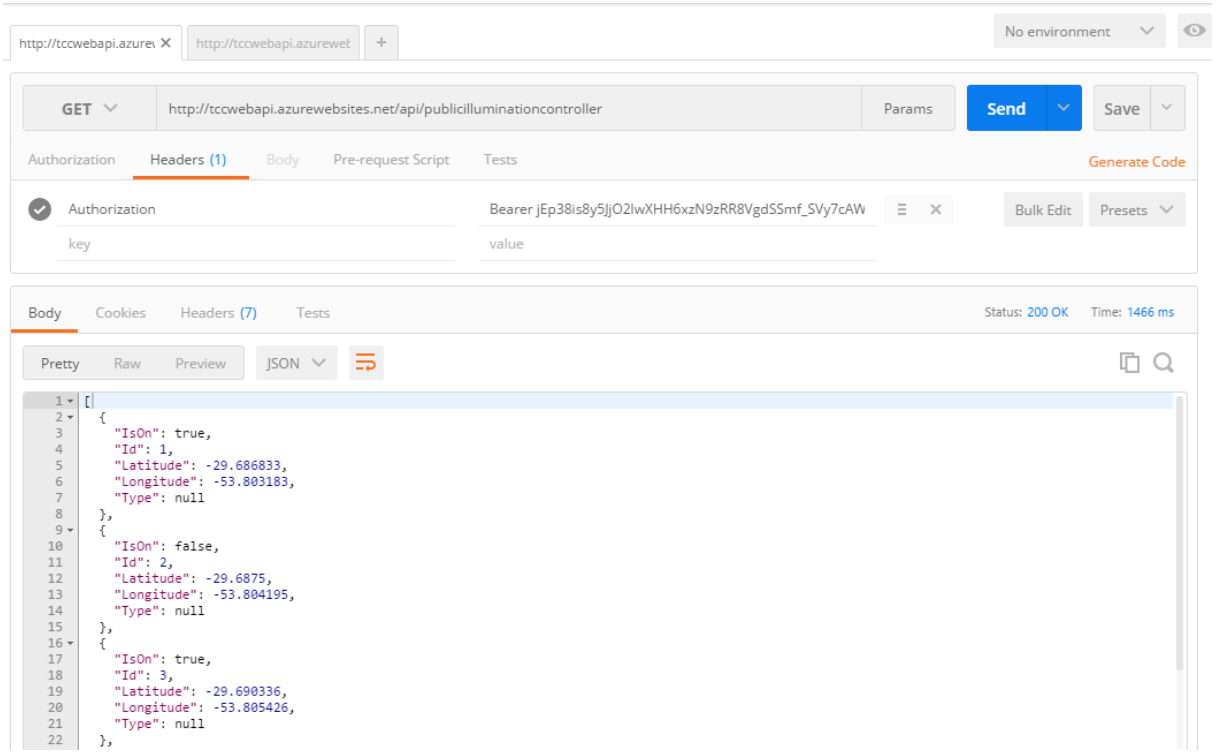
Figura 4.13 – Teste da rota de tokens



Fonte: próprio autor

De posse do token, podemos acessar os dados de cada Tenant, como mostra a **Figura 4.14**.

Figura 4.14 – Teste da rota GET PublicIlluminationController



5 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

O presente projeto apresentou uma arquitetura Multi-Tenant voltada para uma aplicação de Internet das Coisas. A análise realizada a priori indicou as diferentes possibilidades que esse tipo de aplicação traz para o desenvolvedor e seus clientes e como isso pode ser atingido. Por isso esse tipo de arquitetura é tão utilizado em SaaS: as possibilidades vão muito além do software de prateleira, trazendo conceitos importantes para o cliente, como a personalização, até certo ponto, da aplicação para suas necessidades.

A arquitetura Multi-Tenant criada aqui foi aplicada para um projeto de Internet das Coisas. IoT foi a base escolhida para o projeto devido à sua proximidade com a Engenharia de Computação e sua expansão nos últimos, e tendência ainda maior de expansão para os próximos. A inteligência artificial, assunto tão comentado, é um ponto que tem muita relação com IoT e o que pode trazer de benefícios essas tecnologias. O projeto aqui apresentado é um ponto de partida: com o crescimento da IoT, a tendência é que se tenha cada mais equipamento "inteligentes", ou ao menos com a capacidade de fornecer informações. E é para isso que esse projeto se presta.

Dito isso, tem-se que esse projeto pode se tornar um ponto de partida para cidades mais inteligentes. As cidades, em geral, possuem diversos equipamentos que podem ser considerados "coisas": semáforos, lâmpadas de iluminação pública, medidores de velocidades, câmeras, contêineres de lixo, transformadores de energia... Com algum esforço e trabalho todos esses equipamentos podem se reportar a uma central como a criada no projeto, indicando seu status e outras informações relevantes a seu respeito. E, dentro do conceito de Multi-Tenancy, isso pode ser expandido mais de uma cidade (cada uma delas sendo um Tenant). Cada uma com suas necessidades atingidas pelo software.

Além disso, a arquitetura Multi-Tenant criada não se restringe à IoT: com adaptações, ela pode ser utilizada por qualquer aplicação SaaS. Isso só é possível devido a alguns cuidados tomados durante o desenvolvimento, que possibilitaram ao projeto atingir os 4 pilares do desenvolvimento Multi-Tenant:

1. Escalabilidade: a aplicação é escalável. Isso está comprovado pela arquitetura de dados escolhida e o modelo de acesso, garantindo que 1 ou N Tenants tenham acesso ao seu serviço, sem problemas em seus dados.
2. Customização: a customização existe, em um grau pequeno. O grau de customização vai depender muito de cada aplicação e da necessidade de cada cliente, mas foi visto no projeto: o Tenant tem a possibilidade de escolha, durante a subscrição, do tipo de dado que deseja.
3. Confiabilidade: a utilização dos serviços do Microsoft Azure garante a confiabilidade do sistema para quantos Tenants for necessário. Os serviços estão presentes na nuvem, o que garante a possibilidade de alocar mais ou menos recursos em determinados momentos, dependendo do número de acessos.
4. Segurança: a segurança está garantida pela combinação de vários fatores. No caso da subscrição, não há perigo algum de qualquer Tenant ter acesso aos dados do outro. Esses dados estão armazenados em um banco de dados SQL do Azure, com as senhas criptografadas com o algoritmo MD5. Ou seja, mesmo na muito improvável possibilidade de alguém malicioso obter esses dados, não conseguirá decodifica-los. A arquitetura de

dados garante outro ponto dessa segurança, uma vez que, por possuírem *schemas* separados, não há como um Tenant ter acesso aos dados de outro, acidentalmente ou não. Por fim, o acesso à API criada é protegido por *tokens*, que garantem a autenticação para acesso a esses dados via API.

Outro projeto futuro que também se faz necessário é a criação de aplicações para dispositivos móveis. A criação da Web API foi uma preparação a esse trabalho futuro, uma vez que ela pode ser facilmente acessada a partir de clientes Android e iOS, por exemplo. Uma aplicação moderna não sobrevive, atualmente, somente com um cliente web, mesmo que o mesmo seja responsivo. É necessário que haja uma experiência local para ao usuário, em cada dispositivo móvel disponível no mercado. Nesse ponto, encontra-se o Xamarin, ferramenta que permite a utilização de tecnologias Microsoft para criação de aplicações Android e iOS. Utilizando a base que já existe do projeto, é possível criarmos, com pouca dor de cabeça, clientes para as duas principais plataformas móveis.

Assim, foi possível criarmos um projeto que, muito menos importante do que estar finalizado, abre diversas possibilidades de futuro. Dependendo do caminho que deseja ser seguido, estar-se-á entrando em um campo muito promissor do desenvolvimento de software, seja para dar continuidade a uma aplicação voltada para IoT ou trazendo a arquitetura desenvolvida para o lado do SaaS e criando um outro domínio completamente diferente.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] BETTS, D. et al. **Developing Multi-Tenant Applications for the Cloud on Microsoft Windows Azure**. Microsoft, 2012.

[2] PERES, R. **ASP.NET Multi-Tenant Application Succintly**. Syncfusion, 2015. Disponível em: < http://files2.syncfusion.com/Downloads/Ebooks/ASP.NET_Multitenant_Applications_Succinctly.pdf>. Acesso em 14/3/2016.

[3] **IT Glossary**. Gartner. Disponível em: <http://www.gartner.com/it-glossary/multitenancy>. Acesso em 14/3/2016.

[4] Wikipedia. **MVC**. Disponível em: <https://pt.wikipedia.org/wiki/MVC>. Acesso em 14/3/2016.

[5] W3Schools. **ASP.NET Razor Markup**. Disponível em: <http://www.w3schools.com/aspnet/razor_intro.asp>. Acesso em 14/3/2016.

[6] LERMAN, J.; MILLER, R. **Programming Entity Framework Code First**. Sebastopol, CA, EUA. O'Reilly Media, 2012.

[7] SPROTT, D.; WILKES, L. **Understanding Service-Oriented Architecture**. MSDN, Microsoft. January, 2004. Disponível em: < https://msdn.microsoft.com/en-us/library/aa480021.aspx>. Acesso em 14/3/2016.

[8] MILLETT, S.. **Professional ASP.NET Design Patterns**. Indianapolis, IN, EUA; Wiley Publishing Inc., 2010.

[9] JACOBS, D., AULBACH, S.. **Ruminations on Multi-Tenant Databases**. Aachen, Alemanha: BTW, 2007. Disponível em: <http://www.db.in.tum.de/research/publications/conferences/BTW2007-mtd.pdf>. Acesso em 22/6/2016.

[10] CHONG, F., CARRARO, G., WOLTER, R.. **Multi-Tenant Data Architecture**. Microsoft Corporation, June, 2006. Disponível em: <https://msdn.microsoft.com/library/aa479086.aspx#mlttntda_sdss>. Acesso em 22/6/2016.

[11] CASSEMIRO, G. et. al.. **Taurus – Plataforma de IoT Brasileira**. Disponível em: <http://www.embarcados.com.br/taurus-plataforma-de-iot-brasileira/>. Acesso em 20/6/2016.

[12] PIRES, P. et. al.. **Plataformas para a Internet das Coisas**. Minicurso SBRC 2015, Capítulo 3. Disponível em: <http://sbrc2015.ufes.br/wp-content/uploads/Ch3.pdf>. Acesso em 20/6/2016.

[13] GUBBI, J., BUYYA, R. et. al.. **Internet of Things (IoT): A vision, architectural elements, and future directions**. Future Generation Computer Systems, Elsevier, 2012.

[14] STOLTMANN, T.. **Object-Relational Mapping**. Humboldt Universitat, Berlin. Disponível em: <http://www.informatik.hu-

berlin.de/de/forschung/gebiete/rok/wbi/teaching/archive/ws1314/sp_semtext/07_ORM.pdf>. Acesso em 23/6/2016.

[15] ZHANG, J. et. al.. **Study and Realization of Authentication Technique Based on MD5 Algorithm**. CNKI, China, 2004.

[16] CARY, A.. **Authentication and Authorization**. Drexel University, 2012. Disponível em: <https://www.webjunction.org/documents/webjunction/Authentication_and_Authorization.html>. Acesso em 23/6/2016.

[17] BRUEGGER, P.. **Authentication and Authorization**. University of Fribourg. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.89.8612&rep=rep1&type=pdf#page=27>. Acesso em 23/6/2016.

[18] SEM AUTOR. **OWIN – Open Web Interface for .NET**. Disponível em: <http://owin.org/>. Acesso em 23/6/2016.

[19] HEJLSBERG, A., WILTAMUTH, S., GOLDE, P.. **The C# Programming Language**. Boston, MA, EUA; Pearson Education, 2004

[20] FREEMAN, A.. **Pro ASP.NET MVC 5**. Apress Publishing, 2013

[21] SEM AUTOR. **Bing maps for enterprise**. Disponível em: <https://www.microsoft.com/maps/choose-your-bing-maps-API.aspx>. Acesso em 23/6/2016.

[22] MC GRAW, G.. **Software Security**. Upper Sadle River, NJ, EUA; Addison-Wesley, 2006

ANEXOS

ANEXO A – Tenant.cs

```
using Microsoft.AspNet.Identity.EntityFramework;

namespace Domain.Models.Tenants
{
    public class Tenant : IdentityUser
    {
        public string HostName { get; set; }
        public string Type { get; set; }
    }
}
```

ANEXO B – TenantDbContext.cs

```
using Domain.Models.Tenants;
using Microsoft.AspNet.Identity.EntityFramework;

namespace Data.Tenants.Context
{
    public class TenantDbContext : IdentityDbContext<Tenant>
    {
        public TenantDbContext() : base ("TenantDatabase")
        {

        }

        public static TenantDbContext Create()
        {
            return new TenantDbContext();
        }
    }
}
```

ANEXO C – Migration InitialCreate

```
namespace Data.Tenants.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class InitialCreate : DbMigration
    {
        public override void Up()
        {
            CreateTable(
                "dbo.AspNetRoles",
                c => new
                    {
                        Id = c.String(nullable: false, maxLength: 128),
                        Name = c.String(nullable: false, maxLength: 256),
                    })
                .PrimaryKey(t => t.Id)
                .Index(t => t.Name, unique: true, name: "RoleNameIndex");

            CreateTable(
```

```
                "dbo.AspNetUserRoles",
                c => new
                    {
                        UserId = c.String(nullable: false, maxLength: 128),
                        RoleId = c.String(nullable: false, maxLength: 128),
                    })
                .PrimaryKey(t => new { t.UserId, t.RoleId })
                .ForeignKey("dbo.AspNetRoles", t => t.RoleId, cascadeDelete: true)
                .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
                .Index(t => t.UserId)
                .Index(t => t.RoleId);

            CreateTable(
                "dbo.AspNetUsers",
                c => new
                    {
                        Id = c.String(nullable: false, maxLength: 128),
                        Email = c.String(maxLength: 256),
                        EmailConfirmed = c.Boolean(nullable: false),
                        PasswordHash = c.String(),
                        SecurityStamp = c.String(),
                        PhoneNumber = c.String(),
                        PhoneNumberConfirmed = c.Boolean(nullable: false),
                        TwoFactorEnabled = c.Boolean(nullable: false),
                        LockoutEndDateUtc = c.DateTime(),
                        LockoutEnabled = c.Boolean(nullable: false),
                        AccessFailedCount = c.Int(nullable: false),
                        UserName = c.String(nullable: false, maxLength: 256),
                    })
                .PrimaryKey(t => t.Id)
                .Index(t => t.UserName, unique: true, name: "UserNameIndex");

            CreateTable(
                "dbo.AspNetUserClaims",
                c => new
                    {
                        Id = c.Int(nullable: false, identity: true),
                        UserId = c.String(nullable: false, maxLength: 128),
                        ClaimType = c.String(),
                        ClaimValue = c.String(),
                    })
                .PrimaryKey(t => t.Id)
                .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
                .Index(t => t.UserId);

            CreateTable(
                "dbo.AspNetUserLogins",
                c => new
                    {
                        LoginProvider = c.String(nullable: false, maxLength: 128),
                        ProviderKey = c.String(nullable: false, maxLength: 128),
                        UserId = c.String(nullable: false, maxLength: 128),
                    })
                .PrimaryKey(t => new { t.LoginProvider, t.ProviderKey, t.UserId })
                .ForeignKey("dbo.AspNetUsers", t => t.UserId, cascadeDelete: true)
                .Index(t => t.UserId);

        }

        public override void Down()
        {
            DropForeignKey("dbo.AspNetUserRoles", "UserId", "dbo.AspNetUsers");
```

```
            DropForeignKey("dbo.AspNetUserLogins", "UserId", "dbo.AspNetUsers");
            DropForeignKey("dbo.AspNetUserClaims", "UserId", "dbo.AspNetUsers");
            DropForeignKey("dbo.AspNetUserRoles", "RoleId", "dbo.AspNetRoles");
            DropIndex("dbo.AspNetUserLogins", new[] { "UserId" });
            DropIndex("dbo.AspNetUserClaims", new[] { "UserId" });
            DropIndex("dbo.AspNetUsers", "UserNameIndex");
            DropIndex("dbo.AspNetUserRoles", new[] { "RoleId" });
            DropIndex("dbo.AspNetUserRoles", new[] { "UserId" });
            DropIndex("dbo.AspNetRoles", "RoleNameIndex");
            DropTable("dbo.AspNetUserLogins");
            DropTable("dbo.AspNetUserClaims");
            DropTable("dbo.AspNetUsers");
            DropTable("dbo.AspNetUserRoles");
            DropTable("dbo.AspNetRoles");
        }
    }
}
```

ANEXO D – IRepositoryBase.cs

```
using System.Collections.Generic;

namespace Domain.Interfaces.Repositories
{
    /// <summary>
    /// Interface to access the repository.
    /// </summary>
    /// <typeparam name="TEntity"></typeparam>
    public interface IRepositoryBase<TEntity> where TEntity : class
    {
        IEnumerable<TEntity> GetAll();
        TEntity Find(int id);
        void Add(TEntity obj);
        void Update(TEntity obj);
        void Remove(TEntity obj);
        void Dispose();
    }
}
```

ANEXO E – ITenantRepository.cs

```
using Domain.Models.Tenants;

namespace Domain.Interfaces.Repositories
{
    public interface ITenantRepository : IRepositoryBase<Tenant>
    {
        Tenant FindByCredentials(string username, string password);
    }
}
```

ANEXO F – TenantRepository.cs

```
using Data.Tenants.Context;
using Domain.Interfaces.Repositories;
using Domain.Models.Tenants;
using System;
using System.Collections.Generic;
```

```
using System.Data.Entity;
using System.Linq;

namespace Data.Tenants.Repositories
{
    public class TenantRepository : IDisposable, ITenantRepository
    {
        private TenantDbContext context = Create();

        public static TenantDbContext Create()
        {
            return TenantDbContext.Create();
        }

        public void Add(Tenant obj)
        {
            context.Set<Tenant>().Add(obj);
            context.SaveChanges();
        }

        public void Dispose()
        {
            throw new NotImplementedException();
        }

        public Tenant Find(int id)
        {
            return context.Set<Tenant>().Find(id);
        }

        public Tenant FindByCredentials(string username, string password)
        {
            return context.Set<Tenant>().Where(t => t.UserName == username &&
t.PasswordHash == password).FirstOrDefault();
        }

        public IEnumerable<Tenant> GetAll()
        {
            return context.Set<Tenant>().ToList();
        }

        public void Remove(Tenant obj)
        {
            context.Set<Tenant>().Remove(obj);
            context.SaveChanges();
        }

        public void Update(Tenant obj)
        {
            context.Entry(obj).State = EntityState.Modified;
            context.SaveChanges();
        }
    }
}
```

ANEXO G – Thing.cs

```
using System;
using System.ComponentModel.DataAnnotations.Schema;

namespace Domain.Models.Things
```

```
{
    public abstract class Thing
    {
        public int Id { get; set; }
        public decimal Latitude { get; set; }
        public decimal Longitude { get; set; }
        [NotMapped]
        public Type Type { get; set; }
    }
}
```

ANEXO H – ContainerLevelMeasurer.cs

```
namespace Domain.Models.Things
{
    public class ContainerLevelMeasurer : Thing
    {
        public decimal Level { get; set; }
    }
}
```

ANEXO I – PublicIlluminationController.cs

```
namespace Domain.Models.Things
{
    public class PublicIlluminationController : Thing
    {
        public bool IsOn { get; set; }
    }
}
```

ANEXO J – ContainerLevelMeasurerConfiguration.cs

```
using Domain.Models.Things;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity.ModelConfiguration;

namespace Data.IoT.Mapping
{
    public class ContainerLevelMeasurerConfiguration : EntityTypeConfiguration<Con-
tainerLevelMeasurer>
    {
        public ContainerLevelMeasurerConfiguration(string schema)
        {
            ToTable("ContainerLevelMeasurer", schema);

            HasKey(x => x.Id);

            Property(x => x.Id)
                .IsRequired()
                .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);

            Property(x => x.Latitude)
                .IsRequired()
                .HasPrecision(10, 6);

            Property(x => x.Longitude)
                .IsRequired()
```

```
                .HasPrecision(10, 6);

            Property(x => x.Level)
                .HasPrecision(12, 1);
        }
    }
}
```

ANEXO K – PublicIlluminationControllerConfiguration.cs

```
using Domain.Models.Things;
using System.ComponentModel.DataAnnotations.Schema;
using System.Data.Entity.ModelConfiguration;

namespace Data.IoT.Mapping
{
    public class PublicIlluminationControllerConfiguration : EntityTypeConfigura-
tion<PublicIlluminationController>
    {
        public PublicIlluminationControllerConfiguration(string schema)
        {
            ToTable("PublicIlluminationController", schema);

            HasKey(x => x.Id);

            Property(x => x.Id)
                .IsRequired()
                .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);

            Property(x => x.Latitude)
                .IsRequired()
                .HasPrecision(10, 6);

            Property(x => x.Longitude)
                .IsRequired()
                .HasPrecision(10, 6);
        }
    }
}
```

ANEXO L – IoTDataContext.cs

```
using Data.IoT.Mapping;
using Domain.Models.Things;
using System;
using System.Collections.Concurrent;
using System.Data.Common;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;

namespace Data.IoT.Context
{
    public class IoTDataContext : DbContext
    {
        private static ConcurrentDictionary<Tuple<string, string>, DbCompiledModel>
modelCache = new ConcurrentDictionary<Tuple<string, string>, DbCompiledModel>();

        private IoTDataContext(DbConnection connection, DbCompiledModel model) :
base(connection, model, contextOwnsConnection: false)
```

```
        {

        }

        public DbSet<Thing> Things { get; set; }

        public static IoTDataContext Create(string tenantSchema, DbConnection connec-
tion, string thingType)
        {
            Database.SetInitializer<IoTDataContext>(null);
            DbCompiledModel compiledModel = modelCache.GetOrAdd(
                Tuple.Create(connection.ConnectionString, tenantSchema), t => {
                    DbModelBuilder builder = new DbModelBuilder();
                    builder.Conventions.Remove();

                    if (thingType.Contains("ContainerLevel"))
                        builder.Configurations.Add<ContainerLevelMeasurer>(new Con-
tainerLevelMeasurerConfiguration(tenantSchema));
                    else if (thingType.Contains("PublicIllumination"))
                        builder.Configurations.Add<PublicIlluminationController>(new
PublicIlluminationControllerConfiguration(tenantSchema));

                    var model = builder.Build(connection);
                    return model.Compile();
                }
            );

            return new IoTDataContext(connection, compiledModel);
        }

        public static void ProvisionTenant(string tenantSchema, DbConnection connec-
tion, string thingType)
        {
            using (var ctx = Create(tenantSchema, connection, thingType))
            {
                if (!ctx.Database.Exists())
                {
                    ctx.Database.Create();
                }
                else
                {
                    var createScript = ((IObjectContextAdapter)ctx).ObjectContext.Cre-
ateDatabaseScript();
                    ctx.Database.ExecuteSqlCommand(createScript);
                }
            }
        }
    }
}
```

ANEXO M – IThingRepository.cs

```
namespace Domain.Interfaces.Repositories
{
    public interface IThingRepository<TEntity> : IRepositoryBase<TEntity> where
TEntity : class
    {

    }
}
```

ANEXO N – ContainerLevelMeasurerRepository.cs

```
using System;
using System.Collections.Generic;
using Domain.Interfaces.Repositories;
using Domain.Models.Things;
using Data.IoT.Context;
using System.Data.Common;

namespace Data.IoT.Repositories
{
    public class ContainerLevelMeasurerRepository : IThingRepository<Contain-
erLevelMeasurer>
    {
        private IoTDataContext dbContext;

        public ContainerLevelMeasurerRepository(string tenantSchema, DbConnection con-
nection)
        {
            dbContext = IoTDataContext.Create(tenantSchema, connection, "Contain-
erLevel");
        }

        public static ContainerLevelMeasurerRepository Create(string tenantSchema,
DbConnection connection)
        {
            IoTDataContext.ProvisionTenant(tenantSchema, connection, "Contain-
erLevel");
            return new ContainerLevelMeasurerRepository(tenantSchema, connection);
        }

        public static ContainerLevelMeasurerRepository Get(string tenantSchema, DbCon-
nection connection)
        {
            return new ContainerLevelMeasurerRepository(tenantSchema, connection);
        }

        public void Add(ContainerLevelMeasurer obj)
        {
            dbContext.Set<ContainerLevelMeasurer>().Add(obj);
            dbContext.SaveChanges();
        }

        public void Dispose()
        {
            throw new NotImplementedException();
        }

        public ContainerLevelMeasurer Find(int id)
        {
            return dbContext.Set<ContainerLevelMeasurer>().Find(id);
        }

        public IEnumerable<ContainerLevelMeasurer> GetAll()
        {
            return dbContext.Set<ContainerLevelMeasurer>();
        }

        public void Remove(ContainerLevelMeasurer obj)
        {
```

```
            dbContext.Set<ContainerLevelMeasurer>().Remove(obj);
            dbContext.SaveChanges();
        }

        public void Update(ContainerLevelMeasurer obj)
        {
            dbContext.Entry<ContainerLevelMeasurer>(obj).State = System.Data.En-
tity.EntityState.Modified;
            dbContext.SaveChanges();
        }
    }
}
```

ANEXO O – PublicIlluminationControllerRepository.cs

```
using System;
using System.Collections.Generic;
using Domain.Interfaces.Repositories;
using Domain.Models.Things;
using Data.IoT.Context;
using System.Data.Common;

namespace Data.IoT.Repositories
{
    public class PublicIlluminationControllerRepository : IThingRepository<PublicIllu-
minationController>
    {
        private IoTDataContext dbContext;

        public PublicIlluminationControllerRepository(string tenantSchema, DbConnec-
tion connection)
        {
            dbContext = IoTDataContext.Create(tenantSchema, connection, "PublicIllumi-
nation");
        }

        public static PublicIlluminationControllerRepository Create(string tenant-
Schema, DbConnection connection)
        {
            IoTDataContext.ProvisionTenant(tenantSchema, connection, "PublicIllumina-
tion");
            return new PublicIlluminationControllerRepository(tenantSchema, connec-
tion);
        }

        public static PublicIlluminationControllerRepository Get(string tenantSchema,
DbConnection connection)
        {
            return new PublicIlluminationControllerRepository(tenantSchema, connec-
tion);
        }

        public void Add(PublicIlluminationController obj)
        {
            dbContext.Set<PublicIlluminationController>().Add(obj);
            dbContext.SaveChanges();
        }

        public void Dispose()
        {
            throw new NotImplementedException();
```

```
        }

        public PublicIlluminationController Find(int id)
        {
            return dbContext.Set<PublicIlluminationController>().Find(id);
        }

        public IEnumerable<PublicIlluminationController> GetAll()
        {
            return dbContext.Set<PublicIlluminationController>();
        }

        public void Remove(PublicIlluminationController obj)
        {
            dbContext.Set<PublicIlluminationController>().Remove(obj);
            dbContext.SaveChanges();
        }

        public void Update(PublicIlluminationController obj)
        {
            dbContext.Entry<PublicIlluminationController>(obj).State = System.Data.En-
tity.EntityState.Modified;
            dbContext.SaveChanges();
        }
    }
}
```

ANEXO P – IServiceBase.cs

```
using System.Collections.Generic;

namespace Domain.Interfaces.Services
{
    /// <summary>
    /// Interface to access the services.
    /// </summary>
    /// <typeparam name="TEntity"></typeparam>
    public interface IServiceBase<TEntity> where TEntity : class
    {
        IEnumerable<TEntity> GetAll();
        TEntity Find(int id);
        void Add(TEntity obj);
        void Update(TEntity obj);
        void Remove(TEntity obj);
        void Dispose();
    }
}
```

ANEXO Q – IThingService.cs

```
namespace Domain.Interfaces.Services
{
    public interface IThingService<TEntity> : IServiceBase<TEntity> where TEntity :
class
    {
    }
}
```

ANEXO R – IContainerLevelMeasurerService.cs

```
using Domain.Models.Things;

namespace Domain.Interfaces.Services
{
    public interface IContainerLevelMeasurerService : IThingService<Contain-
erLevelMeasurer>
    {
    }
}
```

ANEXO S – IPublicIlluminationControllerService.cs

```
using Domain.Models.Things;

namespace Domain.Interfaces.Services
{
    public interface IPublicIlluminationControllerService : IThingService<PublicIllu-
minationController>
    {
    }
}
```

ANEXO T – ServiceBase.cs

```
using System;
using System.Collections.Generic;
using Domain.Interfaces.Repositories;
using Domain.Interfaces.Services;

namespace Domain.Services
{
    /// <summary>
    /// Base service class to access core data repository.
    /// </summary>
    /// <typeparam name="TEntity"></typeparam>
    public class ServiceBase<TEntity> : IDisposable, IServiceBase<TEntity> where
TEntity : class
    {
        /// <summary>Repository use in this base class.</summary>
        private readonly IRepositoryBase<TEntity> _repository;

        /// <summary>
        /// Constructor.
        /// </summary>
        /// <param name="repository"></param>
        public ServiceBase(IRepositoryBase<TEntity> repository)
        {
            _repository = repository;
        }

        /// <summary>
        /// Get all entites from repository.
        /// </summary>
        /// <returns></returns>
        public IEnumerable<TEntity> GetAll()
        {
            return _repository.GetAll();
```

```
        }

        /// <summary>
        /// Find entity by id.
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public TEntity Find(int id)
        {
            return _repository.Find(id);
        }

        /// <summary>
        /// Add new entity.
        /// </summary>
        /// <param name="obj"></param>
        public void Add(TEntity obj)
        {
            _repository.Add(obj);
        }

        /// <summary>
        /// Update entity.
        /// </summary>
        /// <param name="obj"></param>
        public void Update(TEntity obj)
        {
            _repository.Update(obj);
        }

        /// <summary>
        /// Remove entity.
        /// </summary>
        /// <param name="obj"></param>
        public void Remove(TEntity obj)
        {
            _repository.Remove(obj);
        }

        /// <summary>
        /// Dispose this service.
        /// </summary>
        public void Dispose()
        {
            _repository.Dispose();
        }
    }
}
```

ANEXO U – ThingService.cs

```
using Domain.Interfaces.Repositories;
using Domain.Interfaces.Services;

namespace Domain.Services
{
    public class ThingService<TEntity> : ServiceBase<TEntity>, IThingService<TEntity>
where TEntity : class
    {
        /// <summary>Repository use in this base class.</summary>
        private readonly IThingRepository<TEntity> _repository;
```

```
        /// <summary>
        /// Default constructor.
        /// </summary>
        /// <param name="clienteRepository"></param>
        public ThingService(IThingRepository<TEntity> repository)
            : base(repository)
        {
            _repository = repository;
        }

    }
}
```

ANEXO V – ContainerLevelMeasurerService.cs

```
using Domain.Interfaces.Repositories;
using Domain.Interfaces.Services;
using Domain.Models.Things;

namespace Domain.Services
{
    public class ContainerLevelMeasurerService : ServiceBase<ContainerLevelMeasurer>,
IContainerLevelMeasurerService
    {
        /// <summary>The service used at this app class.</summary>
        private readonly IThingRepository<ContainerLevelMeasurer> _repository;

        /// <summary>
        /// Default constructor.
        /// </summary>
        /// <param name="service"></param>
        public ContainerLevelMeasurerService(IThingRepository<ContainerLevelMeasurer>
repository)
            : base(repository)
        {
            _repository = repository;
        }
    }
}
```

ANEXO W – PublicIlluminationControllerService.cs

```
using Domain.Interfaces.Repositories;
using Domain.Interfaces.Services;
using Domain.Models.Things;

namespace Domain.Services
{
    public class PublicIlluminationControllerService : ServiceBase<PublicIllumination-
Controller>, IPublicIlluminationControllerService
    {
        /// <summary>The service used at this app class.</summary>
        private readonly IThingRepository<PublicIlluminationController> _repository;

        /// <summary>
        /// Default constructor.
        /// </summary>
        /// <param name="service"></param>
```

```
        public PublicIlluminationControllerService(IThingRepository<PublicIllumina-
tionController> repository)
            : base(repository)
        {
            _repository = repository;
        }
    }
}
```

ANEXO X – IAppServiceBase.cs

```
using System;
using System.Collections.Generic;

namespace Application.Services.Interfaces
{
    public interface IAppServiceBase<TEntity> : IDisposable where TEntity : class
    {
        IEnumerable<TEntity> GetAll();
        TEntity Find(int id);
        void Add(TEntity obj);
        void Update(TEntity obj);
        void Remove(TEntity obj);
    }
}
```

ANEXO Y – IThingAppService

```
namespace Application.Services.Interfaces
{
    public interface IThingAppService<TEntity> : IAppServiceBase<TEntity> where
TEntity : class
    {
    }
}
```

ANEXO Z – IContainerLevelMeasurerAppService.cs

```
using Domain.Models.Things;

namespace Application.Services.Interfaces
{
    public interface IContainerLevelMeasurerAppService : IThingAppService<Contain-
erLevelMeasurer>
    {
    }
}
```

ANEXO AA – IPublicIlluminationControllerAppService.cs

```
using Domain.Models.Things;

namespace Application.Services.Interfaces
{
    public interface IPublicIlluminationControllerAppService : IThingAppService<Pub-
licIlluminationController>
```

```
    {
    }
}
```

ANEXO AB – AppServiceBase.cs

```
using Application.Services.Interfaces;
using Domain.Interfaces.Services;
using System;
using System.Collections.Generic;

namespace Application.Services
{
    /// <summary>
    /// App base class implementation to access servers logic.
    /// </summary>
    /// <typeparam name="TEntity"></typeparam>
    public class AppServiceBase<TEntity> : IDisposable, IAppServiceBase<TEntity> where
TEntity : class
    {
        /// <summary>The service base used at this app class.</summary>
        protected readonly IServiceBase<TEntity> _serviceBase;

        /// <summary>
        /// Default constructor.
        /// </summary>
        /// <param name="serviceBase"></param>
        public AppServiceBase(IServiceBase<TEntity> serviceBase)
        {
            _serviceBase = serviceBase;
        }

        /// <summary>
        /// Get all entities from the service.
        /// </summary>
        /// <returns></returns>
        public IEnumerable<TEntity> GetAll()
        {
            return _serviceBase.GetAll();
        }

        /// <summary>
        /// Find entity by id.
        /// </summary>
        /// <param name="id"></param>
        /// <returns></returns>
        public TEntity Find(int id)
        {
            return _serviceBase.Find(id);
        }

        /// <summary>
        /// Add new entity.
        /// </summary>
        /// <param name="obj"></param>
        public void Add(TEntity obj)
        {
            _serviceBase.Add(obj);
        }

        /// <summary>
```

```
        /// Update entity.
        /// </summary>
        /// <param name="obj"></param>
        public void Update(TEntity obj)
        {
            _serviceBase.Update(obj);
        }

        /// <summary>
        /// Remove entity.
        /// </summary>
        /// <param name="obj"></param>
        public void Remove(TEntity obj)
        {
            _serviceBase.Remove(obj);
        }

        /// <summary>
        /// Dispose this server.
        /// </summary>
        public void Dispose()
        {
            _serviceBase.Dispose();
        }
    }
}
```

ANEXO AC – ThingAppService.cs

```
using Application.Services.Interfaces;
using Domain.Interfaces.Services;

namespace Application.Services
{
    public class ThingAppService<TEntity> : AppServiceBase<TEntity>, IThingAppS-
ervice<TEntity> where TEntity : class
    {
        /// <summary>The service used at this app class.</summary>
        private readonly IThingService<TEntity> _service;

        /// <summary>
        /// Default constructor.
        /// </summary>
        /// <param name="service"></param>
        public ThingAppService(IThingService<TEntity> service)
            : base(service)
        {
            _service = service;
        }
    }
}
```

ANEXO AD – ContainerLevelMeasurerAppService.cs

```
using Application.Services.Interfaces;
using Data.IoT.Repositories;
using Domain.Interfaces.Services;
using Domain.Models.Things;
using Domain.Services;
```

```
using System.Data.Common;

namespace Application.Services
{
    public class ContainerLevelMeasurerAppService : ThingAppService<Contain-
erLevelMeasurer>, IContainerLevelMeasurerAppService
    {
        /// <summary>The service used at this app class.</summary>
        private readonly IContainerLevelMeasurerService _service;

        /// <summary>
        /// Default constructor.
        /// </summary>
        /// <param name="service"></param>
        public ContainerLevelMeasurerAppService(IContainerLevelMeasurerService ser-
vice)
            : base(service)
        {
            _service = service;
        }

        /// <summary>
        /// Factory to create a new application.
        /// </summary>
        /// <returns></returns>
        public static IContainerLevelMeasurerAppService Factory(string tenantSchema,
DbConnection connection, bool isCreation)
        {
            ContainerLevelMeasurerRepository repository;
            if (isCreation)
                repository = ContainerLevelMeasurerRepository.Create(tenantSchema,
connection);
            else
                repository = ContainerLevelMeasurerRepository.Get(tenantSchema, con-
nection);
            var service = new ContainerLevelMeasurerService(repository);
            return new ContainerLevelMeasurerAppService(service);
        }
    }
}
```

ANEXO AE – PublicIlluminationControllerAppService.cs

```
using Application.Services.Interfaces;
using Data.IoT.Repositories;
using Domain.Interfaces.Services;
using Domain.Models.Things;
using Domain.Services;
using System.Data.Common;

namespace Application.Services
{
    public class PublicIlluminationControllerAppService : ThingAppService<PublicIllu-
minationController>, IPublicIlluminationControllerAppService
    {
        /// <summary>The service used at this app class.</summary>
        private readonly IPublicIlluminationControllerService _service;

        /// <summary>
        /// Default constructor.
        /// </summary>
```

```
        /// <param name="service"></param>
        public PublicIlluminationControllerAppService(IPublicIlluminationController-
Service service)
            : base(service)
        {
            _service = service;
        }

        /// <summary>
        /// Factory to create a new application.
        /// </summary>
        /// <returns></returns>
        public static IPublicIlluminationControllerAppService Factory(string tenant-
Schema, DbConnection connection, bool isCreation)
        {
            PublicIlluminationControllerRepository repository;
            if (isCreation)
                repository = PublicIlluminationControllerRepository.Create(tenant-
Schema, connection);
            else
                repository = PublicIlluminationControllerRepository.Get(tenantSchema,
connection);
            var service = new PublicIlluminationControllerService(repository);
            return new PublicIlluminationControllerAppService(service);
        }
    }
}
```

ANEXO AF – SimplePasswordHasher.cs

```
using Microsoft.AspNet.Identity;
using System.Security.Cryptography;
using System.Text;

namespace Common.Infrastructure.Cryptography
{
    public class SimplePasswordHasher : IPasswordHasher
    {
        public string HashPassword(string password)
        {
            MD5 md5 = new MD5CryptoServiceProvider();

            md5.ComputeHash(Encoding.ASCII.GetBytes(password));
            byte[] result = md5.Hash;

            StringBuilder strBuilder = new StringBuilder();
            for (int i = 0; i < result.Length; i++)
            {
                strBuilder.Append(result[i].ToString("x2"));
            }

            return strBuilder.ToString();
        }

        public PasswordVerificationResult VerifyHashedPassword(string hashedPassword,
string providedPassword)
        {
            if (hashedPassword.Equals(providedPassword))
                return PasswordVerificationResult.Success;
            else
                return PasswordVerificationResult.Failed;
```

```
        }
    }
}
```

ANEXO AG – SubscriptionTenantManager.cs

```
using Common.Infrastructure.Cryptography;
using Data.Tenants.Repositories;
using Domain.Models.Tenants;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;

namespace Site.Subscription.Infrastructure
{
    public class SubscriptionTenantManager : UserManager<Tenant>
    {
        public SubscriptionTenantManager(IUserStore<Tenant> store) : base(store) { }

        public static SubscriptionTenantManager Create(IdentityFactoryOptions<Sub-
scriptionTenantManager> options, IOwinContext context)
        {
            var userManager = new SubscriptionTenantManager(new UserStore<Tenant>(Ten-
antRepository.Create()));

            userManager.PasswordHasher = new SimplePasswordHasher();

            return userManager;
        }
    }
}
```

ANEXO AH – Startup.cs (Site.Subscription)

```
using Data.Tenants.Repositories;
using Microsoft.Owin;
using Owin;
using Site.Subscription.Infrastructure;

[assembly: OwinStartupAttribute(typeof(Site.Subscription.Startup))]
namespace Site.Subscription
{
    public class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            app.CreatePerOwinContext(TenantRepository.Create);
            app.CreatePerOwinContext<SubscriptionTenantManager>(SubscriptionTenantMan-
ager.Create);
        }
    }
}
```

ANEXO AI – TenantViewModel.cs

```
using System.ComponentModel.DataAnnotations;
```

```
namespace Site.Subscription.Models
{
    public class TenantViewModel
    {
        [Required]
        [Display(Name = "Username")]
        public string Username { get; set; }

        [Required]
        [EmailAddress]
        [Display(Name = "Email")]
        public string Email { get; set; }

        [Required]
        [StringLength(100, ErrorMessage = "The {0} must be at least {2} characters
long.", MinimumLength = 6)]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [DataType(DataType.Password)]
        [Display(Name = "Confirm password")]
        [Compare("Password", ErrorMessage = "The password and confirmation password do
not match.")]
        public string ConfirmPassword { get; set; }

        public ThingType Type { get; set; }
    }

    public enum ThingType
    {
        ContainerLevelMeasurer,
        PublicIlluminationController
    }
}
```

ANEXO AJ – CreateUser.cshtml

```
@model Site.Subscription.Models.TenantViewModel

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Create User</title>
    <link href="~/Content/bootstrap.min.css" type="text/css" rel="stylesheet" />
</head>
<body>
    <div class="container container-fluid">
        @using (Html.BeginForm())
        {
            @Html.AntiForgeryToken()

            <div class="form-horizontal">
                <h4>Subscribe</h4>
                <hr />
```

```
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(model => model.Username, htmlAttributes: new
{ @class = "control-label col-md-2" })
                    <div class="col-md-10">
                        @Html.EditorFor(model => model.Username, new { htmlAttributes
= new { @class = "form-control" } })
                        @Html.ValidationMessageFor(model => model.Username, "", new
{ @class = "text-danger" })
                    </div>
                </div>

                <div class="form-group">
                    @Html.LabelFor(model => model.Email, htmlAttributes: new { @class
= "control-label col-md-2" })
                    <div class="col-md-10">
                        @Html.EditorFor(model => model.Email, new { htmlAttributes =
new { @class = "form-control" } })
                        @Html.ValidationMessageFor(model => model.Email, "", new
{ @class = "text-danger" })
                    </div>
                </div>

                <div class="form-group">
                    @Html.LabelFor(model => model.Password, htmlAttributes: new
{ @class = "control-label col-md-2" })
                    <div class="col-md-10">
                        @Html.EditorFor(model => model.Password, new { htmlAttributes
= new { @class = "form-control" } })
                        @Html.ValidationMessageFor(model => model.Password, "", new
{ @class = "text-danger" })
                    </div>
                </div>

                <div class="form-group">
                    @Html.LabelFor(model => model.ConfirmPassword, htmlAttributes: new
{ @class = "control-label col-md-2" })
                    <div class="col-md-10">
                        @Html.EditorFor(model => model.ConfirmPassword, new { htmlAt-
tributes = new { @class = "form-control" } })
                        @Html.ValidationMessageFor(model => model.ConfirmPassword, "",
new { @class = "text-danger" })
                    </div>
                </div>

                <div class="form-group">
                    @Html.LabelFor(model => model.Type, htmlAttributes: new { @class =
"control-label col-md-2" })
                    <div class="col-md-10">
                        @Html.DropDownListFor(model => model.Type, EnumHelper.Get-
SelectList(typeof(Site.Subscription.Models.ThingType)), new { @class = "form-con-
trol" })
                        @Html.ValidationMessageFor(model => model.Type, "", new
{ @class = "text-danger" })
                    </div>
                </div>

                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <input type="submit" value="Subscribe!" class="btn btn-de-
fault" />
                    </div>
```

```
                </div>
            </div>
        }
    </div>
</body>
</html>
```

ANEXO AK – AccountController.cs (Site.Subscription)

```
using Domain.Interfaces.Repositories;
using Domain.Models.Tenants;
using Site.Subscription.Models;
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity.Owin;
using Site.Subscription.Infrastructure;
using Common.Infrastructure.Cryptography;
using Microsoft.AspNet.Identity;
using Data.IoT.Repositories;
using System.Data.SqlClient;
using System.Configuration;
using Domain.Models.Things;

namespace Site.Subscription.Controllers
{
    public class AccountController : Controller
    {
        private IThingRepository<ContainerLevelMeasurer> _clmRepository;
        private IThingRepository<PublicIlluminationController> _picRepository;

        private SubscriptionTenantManager _tenantManager;

        public AccountController()
        {
        }

        public AccountController(SubscriptionTenantManager tenantManager)
        {
            TenantManager = tenantManager;
        }

        public SubscriptionTenantManager TenantManager
        {
            get
            {
                return _tenantManager ?? HttpContext.GetOwinContext().GetUserMan-
ager<SubscriptionTenantManager>();
            }
            private set
            {
                _tenantManager = value;
            }
        }

        [AllowAnonymous]
        public ActionResult CreateUser()
        {
            return View();
        }

        [HttpPost]
```

```
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async System.Threading.Tasks.Task<ActionResult> CreateUser(TenantView-
Model model)
        {
            if (ModelState.IsValid)
            {
                var hasher = new SimplePasswordHasher();
                var tenant = new Tenant() { UserName = model.Username, Email =
model.Email, Type = model.Type.ToString() };
                var result = await TenantManager.CreateAsync(tenant, model.Password);
                if (result.Succeeded)
                {
                    switch(model.Type)
                    {
                        case ThingType.ContainerLevelMeasurer:
                            _clmRepository = ContainerLevelMeasurerRepository.Cre-
ate(tenant.Id.ToString(), new SqlConnection(ConfigurationManager.Connection-
Strings["IoTDatabase"].ConnectionString));
                            break;
                        case ThingType.PublicIlluminationController:
                            _picRepository = PublicIlluminationControllerReposi-
tory.Create(tenant.Id.ToString(), new SqlConnection(ConfigurationManager.Connection-
Strings["IoTDatabase"].ConnectionString));
                            break;
                        default:
                            break;
                    }

                    return Redirect("http://tccapplication.azurewebsites.net/");
                }
                AddErrors(result);
            }

            return View(model);
        }

        #region Helpers
        private void AddErrors(IdentityResult result)
        {
            foreach (var error in result.Errors)
            {
                ModelState.AddModelError("", error);
            }
        }
        #endregion
    }
}
```

ANEXO AL – ApplicationTenantManager.cs

```
using Common.Infrastructure.Cryptography;
using Data.Tenants.Repositories;
using Domain.Models.Tenants;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.EntityFramework;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;

namespace Site.Application.Infrastructure
{
```

```
    public class ApplicationTenantManager : UserManager<Tenant>
    {
        public ApplicationTenantManager(IUserStore<Tenant> store) : base(store) { }

        public static ApplicationTenantManager Create(IdentityFactoryOptions<Applica-
tionTenantManager> options, IOwinContext context)
        {
            var userManager = new ApplicationTenantManager(new UserStore<Tenant>(Ten-
antRepository.Create()));
            userManager.PasswordHasher = new SimplePasswordHasher();
            return userManager;
        }
    }
}
```

ANEXO AM – ApplicationSignInManager.cs

```
using Domain.Models.Tenants;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin;
using Microsoft.Owin.Security;

namespace Site.Application.Infrastructure
{
    public class ApplicationSignInManager : SignInManager<Tenant, string>
    {
        public ApplicationSignInManager(ApplicationTenantManager userManager, IAuthen-
ticationManager authenticationManager) : base(userManager, authenticationManager) { }

        public static ApplicationSignInManager Create(IdentityFactoryOptions<Applica-
tionSignInManager> option, IOwinContext context)
        {
            var manager = context.GetUserManager<ApplicationTenantManager>();
            var sign = new ApplicationSignInManager(manager, context.Authentication);
            return sign;
        }
    }
}
```

ANEXO AN – Startup.cs (Site.Application)

```
using Microsoft.Owin;
using Owin;

[assembly: OwinStartupAttribute(typeof(Site.Application.Startup))]
namespace Site.Application
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            ConfigureAuth(app);
        }
    }
}
```

ANEXO AO – Startup.Auth.cs

```
using System;
using Microsoft.AspNet.Identity;
using Microsoft.Owin;
using Microsoft.Owin.Security.Cookies;
using Owin;
using Data.Tenants.Repositories;
using Site.Application.Infrastructure;

namespace Site.Application
{
    public partial class Startup
    {
        public void ConfigureAuth(IAppBuilder app)
        {
            app.CreatePerOwinContext(TenantRepository.Create);
            app.CreatePerOwinContext<ApplicationTenantManager>(ApplicationTenantMan-
ager.Create);
            app.CreatePerOwinContext<ApplicationSignInManager>(ApplicationSignInMan-
ager.Create);
            app.UseCookieAuthentication(new CookieAuthenticationOptions
            {
                AuthenticationType = DefaultAuthenticationTypes.ApplicationCookie,
                LoginPath = new PathString("/Account/Login"),
                CookieName = "TenantLogin",
                CookiePath = "/",
                ExpireTimeSpan = TimeSpan.FromHours(12)
            });
        }
    }
}
```

ANEXO AP – AccountViewModels.cs

```
using System.ComponentModel.DataAnnotations;

namespace Site.Application.Models
{
    public class LoginViewModel
    {
        [Required]
        [Display(Name = "Username")]
        public string Username { get; set; }

        [Required]
        [DataType(DataType.Password)]
        [Display(Name = "Password")]
        public string Password { get; set; }

        [Display(Name = "Remember me?")]
        public bool RememberMe { get; set; }
    }
}
```

ANEXO AQ – Login.cshtml

```
@using Site.Application.Models
@model LoginViewModel
@{
    ViewBag.Title = "Log in";
}
```

```
<h2>@ViewBag.Title</h2>
<div class="row">
    <div class="col-md-8">
        <section id="loginForm">
            @using (Html.BeginForm("Login", "Account", new { ReturnUrl = ViewBag.Re-
turnUrl }, FormMethod.Post, new { @class = "form-horizontal", role = "form" }))
            {
                @Html.AntiForgeryToken()
                <h4>Use a local account to log in.</h4>
                <hr />
                @Html.ValidationSummary(true, "", new { @class = "text-danger" })
                <div class="form-group">
                    @Html.LabelFor(m => m.Username, new { @class = "col-md-2 control-
label" })
                    <div class="col-md-10">
                        @Html.TextBoxFor(m => m.Username, new { @class = "form-con-
trol" })
                        @Html.ValidationMessageFor(m => m.Username, "", new { @class =
"text-danger" })
                    </div>
                </div>
                <div class="form-group">
                    @Html.LabelFor(m => m.Password, new { @class = "col-md-2 control-
label" })
                    <div class="col-md-10">
                        @Html.PasswordFor(m => m.Password, new { @class = "form-con-
trol" })
                        @Html.ValidationMessageFor(m => m.Password, "", new { @class =
"text-danger" })
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <div class="checkbox">
                            @Html.CheckBoxFor(m => m.RememberMe)
                            @Html.LabelFor(m => m.RememberMe)
                        </div>
                    </div>
                </div>
                <div class="form-group">
                    <div class="col-md-offset-2 col-md-10">
                        <input type="submit" value="Log in" class="btn btn-default" />
                    </div>
                </div>
                <p>
                    <a href="http://tccsubcription.azurewebsites.net/">Register as a
new user</a>
                </p>
            }
        </section>
    </div>
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

ANEXO AR – AccountController.cs (Site.Application)

```
using System.Threading.Tasks;
```

```
using System.Web;
using System.Web.Mvc;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Microsoft.Owin.Security;
using Site.Application.Infrastructure;
using Common.Infrastructure.Cryptography;
using Site.Application.Models;

namespace Site.Application.Controllers
{
    [Authorize]
    public class AccountController : Controller
    {
        private ApplicationSignInManager _signInManager;
        private ApplicationTenantManager _tenantManager;

        public AccountController()
        {
        }

        public AccountController(ApplicationTenantManager tenantManager, Application-
SignInManager signInManager )
        {
            TenantManager = tenantManager;
            SignInManager = signInManager;
        }

        public ApplicationSignInManager SignInManager
        {
            get
            {
                return _signInManager ?? HttpContext.GetOwinContext().Get<Application-
SignInManager>();
            }
            private set
            {
                _signInManager = value;
            }
        }

        public ApplicationTenantManager TenantManager
        {
            get
            {
                return _tenantManager ?? HttpContext.GetOwinContext().GetUserMan-
ager<ApplicationTenantManager>();
            }
            private set
            {
                _tenantManager = value;
            }
        }

        //
        // GET: /Account/Login
        [AllowAnonymous]
        public ActionResult Login(string returnUrl)
        {
            ViewBag.ReturnUrl = returnUrl;
            return View();
        }
```

```
        //
        // POST: /Account/Login
        [HttpPost]
        [AllowAnonymous]
        [ValidateAntiForgeryToken]
        public async Task<ActionResult> Login(LoginViewModel model, string returnUrl)
        {
            if (!ModelState.IsValid)
            {
                return View(model);
            }

            var hasher = new SimplePasswordHasher();
            var result = await SignInManager.PasswordSignInAsync(model.Username,
hasher.HashPassword(model.Password), model.RememberMe, shouldLockout: false);
            switch (result)
            {
                case SignInStatus.Success:
                    return RedirectToLocal(returnUrl);
                case SignInStatus.LockedOut:
                    return View("Lockout");
                case SignInStatus.RequiresVerification:
                case SignInStatus.Failure:
                default:
                    ModelState.AddModelError("", "Invalid login attempt.");
                    return View(model);
            }
        }

        //
        // POST: /Account/LogOff
        [HttpPost]
        [ValidateAntiForgeryToken]
        public ActionResult LogOff()
        {
            AuthenticationManager.SignOut(DefaultAuthenticationTypes.Application-
Cookie);
            return RedirectToAction("Index", "Home");
        }

        protected override void Dispose(bool disposing)
        {
            if (disposing)
            {
                if (_tenantManager != null)
                {
                    _tenantManager.Dispose();
                    _tenantManager = null;
                }

                if (_signInManager != null)
                {
                    _signInManager.Dispose();
                    _signInManager = null;
                }
            }

            base.Dispose(disposing);
        }

        #region Helpers
```

```
        private IAuthenticationManager AuthenticationManager
        {
            get
            {
                return HttpContext.GetOwinContext().Authentication;
            }
        }

        private ActionResult RedirectToLocal(string returnUrl)
        {
            if (Url.IsLocalUrl(returnUrl))
            {
                return Redirect(returnUrl);
            }
            return RedirectToAction("Index", "Home");
        }
        #endregion
    }
}
```

ANEXO AS – HomeController.cs

```
using System.Web.Mvc;

namespace Site.Application.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        public ActionResult About()
        {
            ViewBag.Message = "Your application description page.";

            return View();
        }

        public ActionResult Contact()
        {
            ViewBag.Message = "Your contact page.";

            return View();
        }
    }
}
```

ANEXO AT – About.cshtml

```
@{
    ViewBag.Title = "About";
}
<h2>@ViewBag.Title.</h2>
<h3>DESENVOLVIMENTO DE ARQUITETURA MULTI-TENANT PARA INTERNET DAS COISAS</h3>

<p>Trabalho de Graduação apresentado ao curso de Engenharia de Computação da Universi-
dade Federal de Santa Maria (UFSM, RS) como requisito parcial para obtenção do grau de
Engenheiro de Computação</p>
<p>Desenvolvido por Henrique Machado Gasparotto</p>
<p>Orientador: Prof. Dr. Carlos Henrique Barriquello</p>
```

ANEXO AU – Contact.cshtml

```
@{
    ViewBag.Title = "Contact";
}
<h2>@ViewBag.Title.</h2>
<h3>@ViewBag.Message</h3>

<address>
    <strong>E-mail:</strong>   <a href="mailto:hmgasparotto@hotmail.com">hmgaspa-
rotto@hotmail.com</a><br />
</address>
```

ANEXO AV – ThingController.cs

```
using Application.Services;
using Application.Services.Interfaces;
using Domain.Models.Tenants;
using Domain.Models.Things;
using Microsoft.AspNet.Identity;
using Microsoft.AspNet.Identity.Owin;
using Site.Application.Infrastructure;
using System.Configuration;
using System.Data.SqlClient;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;

namespace Site.Application.Controllers
{
    [Authorize]
    public class ThingController : Controller
    {
        private IContainerLevelMeasurerAppService _clmService;
        private IPublicIlluminationControllerAppService _picService;

        private ApplicationTenantManager _tenantManager;
        private Tenant _currentTenant;

        public ApplicationTenantManager TenantManager
        {
            get
            {
                return _tenantManager ?? HttpContext.GetOwinContext().GetUserMan-
ager<ApplicationTenantManager>();
            }
            private set
            {
                _tenantManager = value;
            }
        }

        // GET: Thing
        public ActionResult Index()
        {
            RepositoryGetter();
            if (_currentTenant.Type.Contains("ContainerLevel"))
            {
                return View("IndexContainerLevelMeasurer", _clmService.GetAll());
```

```
            }
            else if (_currentTenant.Type.Contains("PublicIllumination"))
            {
                return View("IndexPublicIlluminationController", _picService.Get-
All());
            }
            return View();
        }

        #region ContainerLevelMeasurer
        public ActionResult CreateContainerLevelMeasurer()
        {
            RepositoryGetter();
            return View();
        }

        [HttpPost]
        public ActionResult CreateContainerLevelMeasurer(ContainerLevelMeasurer model)
        {
            RepositoryGetter();
            if (ModelState.IsValid)
            {
                _clmService.Add(model);
                return RedirectToAction("Index");
            }
            else
            {
                return View(model);
            }
        }

        public ActionResult EditContainerLevelMeasurer(int id)
        {
            RepositoryGetter();
            return View(_clmService.Find(id));
        }

        [HttpPost]
        public ActionResult EditContainerLevelMeasurer(ContainerLevelMeasurer model)
        {
            RepositoryGetter();
            if (ModelState.IsValid)
            {
                _clmService.Add(model);
                return RedirectToAction("Index");
            }
            else
            {
                return View(model);
            }
        }

        public ActionResult DetailsContainerLevelMeasurer(int id)
        {
            RepositoryGetter();
            return View(_clmService.Find(id));
        }

        public ActionResult DeleteContainerLevelMeasurer(int id)
        {
            RepositoryGetter();
            return View(_clmService.Find(id));
```

```
        }

        [HttpPost]
        public ActionResult DeleteContainerLevelMeasurer(ContainerLevelMeasurer model)
        {
            RepositoryGetter();
            if (ModelState.IsValid)
            {
                _clmService.Remove(model);
                return RedirectToAction("Index");
            }
            else
            {
                return View(model);
            }
        }
        #endregion

        #region PublicIlluminationController
        public ActionResult CreatePublicIlluminationController()
        {
            RepositoryGetter();
            return View();
        }

        [HttpPost]
        public ActionResult CreatePublicIlluminationController(PublicIlluminationCon-
troller model)
        {
            RepositoryGetter();
            if (ModelState.IsValid)
            {
                _picService.Add(model);
                return RedirectToAction("Index");
            }
            else
            {
                return View(model);
            }
        }

        public ActionResult EditPublicIlluminationController(int id)
        {
            RepositoryGetter();
            return View(_picService.Find(id));
        }

        [HttpPost]
        public ActionResult EditPublicIlluminationController(PublicIlluminationCon-
troller model)
        {
            RepositoryGetter();
            if (ModelState.IsValid)
            {
                _picService.Add(model);
                return RedirectToAction("Index");
            }
            else
            {
                return View(model);
            }
        }
```

```
        public ActionResult DetailsPublicIlluminationController(int id)
        {
            RepositoryGetter();
            return View(_picService.Find(id));
        }

        public ActionResult DeletePublicIlluminationController(int id)
        {
            RepositoryGetter();
            return View(_picService.Find(id));
        }

        [HttpPost]
        public ActionResult DeletePublicIlluminationController(PublicIlluminationCon-
troller model)
        {
            RepositoryGetter();
            if (ModelState.IsValid)
            {
                _picService.Remove(model);
                return RedirectToAction("Index");
            }
            else
            {
                return View(model);
            }
        }
        #endregion

        #region Helpers
        public void RepositoryGetter()
        {
            if ((HttpContext != null) && (_clmService == null) & (_picService ==
null))
            {
                var tenant = Task.Run(() => TenantManager.FindByIdAsync(HttpCon-
text.User.Identity.GetUserId()));
                _currentTenant = tenant.Result;
                if (_currentTenant.Type.Contains("ContainerLevel"))
                {
                    _clmService = ContainerLevelMeasurerAppService.Factory(_cur-
rentTenant.Id, new SqlConnection(ConfigurationManager.ConnectionStrings["IoTData-
base"].ConnectionString), false);
                }
                else if (_currentTenant.Type.Contains("PublicIllumination"))
                {
                    _picService = PublicIlluminationControllerAppService.Factory(_cur-
rentTenant.Id, new SqlConnection(ConfigurationManager.ConnectionStrings["IoTData-
base"].ConnectionString), false);
                }
            }
        }
        #endregion
    }
}
```

ANEXO AW – EditContainerLevelMeasurer.cshtml

```
@model Domain.Models.Things.ContainerLevelMeasurer
```

```
@{
    ViewBag.Title = "Edit Container Level Measurer";
}

<h2>Edit Container Level Measurer</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Container Level Measurer</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.Id)

        <div class="form-group">
            @Html.LabelFor(model => model.Latitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Latitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Latitude, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Longitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Longitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Longitude, "", new { @class
= "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Level, htmlAttributes: new { @class = "con-
trol-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Level, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Level, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>
```

```
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

ANEXO AX – EditPublicIlluminationController.cshtml

```
@model Domain.Models.Things.PublicIlluminationController

@{
    ViewBag.Title = "Edit Public Illumination Controller";
}

<h2>Edit Public Illumination Controller</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Public Illumination Controller</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.Id)

        <div class="form-group">
            @Html.LabelFor(model => model.Latitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Latitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Latitude, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Longitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Longitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Longitude, "", new { @class
= "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.IsOn, htmlAttributes: new { @class = "con-
trol-label col-md-2" })
            <div class="col-md-10">
                <div class="checkbox">
                    @Html.EditorFor(model => model.IsOn)
                    @Html.ValidationMessageFor(model => model.IsOn, "", new { @class =
"text-danger" })
                </div>
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
```

```
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

ANEXO AY – CreateContainerLevelMeasurer.cshtml

```
@model Domain.Models.Things.ContainerLevelMeasurer

@{
    ViewBag.Title = "Create Container Level Measurer";
}

<h2>Create Container Level Measurer</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Container Level Measurer</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Latitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Latitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Latitude, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Longitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Longitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Longitude, "", new { @class
= "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Level, htmlAttributes: new { @class = "con-
trol-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Level, new { htmlAttributes = new
{ @class = "form-control" } })
```

```
                @Html.ValidationMessageFor(model => model.Level, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

ANEXO AZ – CreatePublicIlluminationController.cshtml

```
@model Domain.Models.Things.PublicIlluminationController

@{
    ViewBag.Title = "Create Public Illumination Controller";
}

<h2>Create Public Illumination Controller</h2>


@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Public Illumination Controller</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        <div class="form-group">
            @Html.LabelFor(model => model.Latitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Latitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Latitude, "", new { @class =
"text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Longitude, htmlAttributes: new { @class =
"control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Longitude, new { htmlAttributes = new
{ @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Longitude, "", new { @class
= "text-danger" })
            </div>
        </div>
```

```
        <div class="form-group">
            @Html.LabelFor(model => model.IsOn, htmlAttributes: new { @class = "con-
trol-label col-md-2" })
            <div class="col-md-10">
                <div class="checkbox">
                    @Html.EditorFor(model => model.IsOn)
                    @Html.ValidationMessageFor(model => model.IsOn, "", new { @class =
"text-danger" })
                </div>
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}

<div>
    @Html.ActionLink("Back to List", "Index")
</div>

@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

ANEXO BA – DetailsContainerLevelMeasurer.cshtml

```
@model Domain.Models.Things.ContainerLevelMeasurer

@{
    ViewBag.Title = "Container Level Measurer Details";
}

<h2>Container Level Measurer Details</h2>

<div>
    <h4>Container Level Measurer</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Latitude)
        </dt>

        <dd>
            @Model.Latitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Longitude)
        </dt>

        <dd>
            @Model.Longitude.ToString("0.000000")
        </dd>

        <dt>
```

```
            @Html.DisplayNameFor(model => model.Level)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Level)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "EditContainerLevelMeasurer", new { id = Model.Id }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

ANEXO BB – DetailsPublicIlluminationController.cshtml

```
@model Domain.Models.Things.PublicIlluminationController

@{
    ViewBag.Title = "Public Illumination Controller Details";
}

<h2>Public Illumination Controller Details</h2>

<div>
    <h4>Public Illumination Controller</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Latitude)
        </dt>

        <dd>
            @Model.Latitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Longitude)
        </dt>

        <dd>
            @Model.Longitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.IsOn)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.IsOn)
        </dd>

    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "EditPublicIlluminationController", new { id =
Model.Id }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

ANEXO BC – DeleteContainerLevelMeasurer.cshtml

```
@model Domain.Models.Things.ContainerLevelMeasurer

@{
    ViewBag.Title = "Delete Container Level Measurer";
}

<h2>Delete Container Level Measurer</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Container Level Measurer</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Latitude)
        </dt>

        <dd>
            @Model.Latitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Longitude)
        </dt>

        <dd>
            @Model.Longitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Level)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.Level)
        </dd>

    </dl>

    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()

        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

ANEXO BD – DeletePublicIlluminationController.cshtml

```
@model Domain.Models.Things.PublicIlluminationController

@{
    ViewBag.Title = "Delete Public Illumination Controller";
}
```

```
<h2>Delete Public Illumination Controller</h2>

<h3>Are you sure you want to delete this?</h3>
<div>
    <h4>Public Illumination Controller</h4>
    <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Latitude)
        </dt>

        <dd>
            @Model.Latitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.Longitude)
        </dt>

        <dd>
            @Model.Longitude.ToString("0.000000")
        </dd>

        <dt>
            @Html.DisplayNameFor(model => model.IsOn)
        </dt>

        <dd>
            @Html.DisplayFor(model => model.IsOn)
        </dd>

    </dl>

    @using (Html.BeginForm()) {
        @Html.AntiForgeryToken()

        <div class="form-actions no-color">
            <input type="submit" value="Delete" class="btn btn-default" /> |
            @Html.ActionLink("Back to List", "Index")
        </div>
    }
</div>
```

ANEXO BE – IndexContainerLevelMeasurer.cshtml

```
@model IEnumerable<Domain.Models.Things.ContainerLevelMeasurer>

@{
    ViewBag.Title = "Container Level Measurer";
}

@section scripts
{
    <script src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
    <script>
        var map;
        function loadMapScenario() {
            map = new Microsoft.Maps.Map(document.getElementById('mapViewer'), {
                credentials: 'AhTFkkLNWCZeF1-u8TmOV-
GfumUUQZx6fvbEydcnpeOl2z6HkE09oXUayXjV6yW4V',
```

```
                center: new Microsoft.Maps.Location(-29.691037,-53.835232),
                zoom: 13
            });
            Microsoft.Maps.Events.addHandler(map, 'click', function (e) {
                $("#dialog-confirm").dialog({
                    resizable: false,
                    height:140,
                    modal: true,
                    buttons: {
                        "Create new": function() {
                            var point = new Microsoft.Maps.Point(e.getX(), e.getY());
                            var loc = e.target.tryPixelToLocation(point);
                            var location = new Microsoft.Maps.Location(loc.latitude,
loc.longitude);
                            var ContainerLevelMeasurer =
                            {
                                "Latitude": location.latitude,
                                "Longitude": location.longitude,
                                "Level": 0
                            };
                            $.ajax({
                                url: '/Thing/CreateContainerLevelMeasurer/',
                                data: JSON.stringify(ContainerLevelMeasurer),
                                type: 'POST',
                                contentType: 'application/json; charset=utf-8'
                            });
                            window.location.href = "@Url.Action("Index")";
                            $(this).dialog( "close" );
                        },
                        Cancel: function() {
                            $(this).dialog( "close" );
                        }
                    }
                });
            });
            @foreach (var location in Model)
            {
                <text>
                var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Loca-
tion(@location.Latitude, @location.Longitude), { icon: @{ if (location.Level > 0.2m)
{ <text>'Content/Images/trash.png'</text> } else { <text>'Content/Images/trash-low-
level.png'</text> } } });
                map.entities.push(pushpin);
                Microsoft.Maps.Events.addHandler(pushpin, 'click', function () {
                    window.location.href = "@Url.Action("DetailsContainerLevelMeas-
urer", new { Id = location.Id })";
                });
                </text>
            }
        }
    </script>
    <script src="http://www.bing.com/api/maps/mapcontrol?branch=re-
lease&callback=loadMapScenario" async defer></script>
}

<link rel="stylesheet" href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-
ui.css">

<div style="display: none;" id="dialog-confirm" title="Create new Container Level
Measurer?">
    <p><span class="ui-icon ui-icon-alert" style="float:left; margin:0 7px 20px
0;"></span>Are you sure?</p>
```

```
</div>

<h2>Container Level Measurer</h2>

<p>
    @Html.ActionLink("Create New", "CreateContainerLevelMeasurer")
</p>

<div id='mapViewer' style='width: 100%; height: 300px;'></div>

<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Latitude)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Longitude)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Level)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @item.Latitude.ToString("0.000000")
        </td>
        <td>
            @item.Longitude.ToString("0.000000")
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Level)
        </td>
        <td>
            @Html.ActionLink("Edit", "EditContainerLevelMeasurer", new { id=item.Id })
|
            @Html.ActionLink("Details", "DetailsContainerLevelMeasurer", new
{ id=item.Id }) |
            @Html.ActionLink("Delete", "DeleteContainerLevelMeasurer", new
{ id=item.Id })
        </td>
    </tr>
}

</table>
```

ANEXO BF – IndexPublicIlluminationController

```
@model IEnumerable<Domain.Models.Things.PublicIlluminationController>

@{
    ViewBag.Title = "Public Illumination Controller";
}

@section scripts
{
    <script src="//code.jquery.com/ui/1.11.4/jquery-ui.js"></script>
    <script>
```

```
        var map;
        function loadMapScenario() {
            map = new Microsoft.Maps.Map(document.getElementById('mapViewer'), {
                credentials: 'AhTFkkLNWCZeF1-u8TmOV-
GfumUUQZx6fvbEydcnpeOl2z6HkE09oXUayXjV6yW4V',
                center: new Microsoft.Maps.Location(-29.691037,-53.835232),
                zoom: 13
            });
            Microsoft.Maps.Events.addHandler(map, 'click', function (e) {
                $("#dialog-confirm").dialog({
                    resizable: false,
                    height:140,
                    modal: true,
                    buttons: {
                        "Create new": function() {
                            var point = new Microsoft.Maps.Point(e.getX(), e.getY());
                            var loc = e.target.tryPixelToLocation(point);
                            var location = new Microsoft.Maps.Location(loc.latitude,
loc.longitude);
                            var PublicIlluminationController =
                            {
                                "Latitude": location.latitude,
                                "Longitude": location.longitude,
                                "IsOn": true
                            };
                            $.ajax({
                                url: '/Thing/CreatePublicIlluminationController/',
                                data: JSON.stringify(PublicIlluminationController),
                                type: 'POST',
                                contentType: 'application/json; charset=utf-8'
                            });
                            window.location.href = "@Url.Action("Index")";
                            $(this).dialog( "close" );
                        },
                        Cancel: function() {
                            $(this).dialog( "close" );
                        }
                    }
                });
            });
            @foreach (var location in Model)
            {
                <text>
                var pushpin = new Microsoft.Maps.Pushpin(new Microsoft.Maps.Loca-
tion(@location.Latitude, @location.Longitude), { icon: @{ if (location.IsOn)
{ <text>'/Content/Images/light.png'</text> } else { <text>'/Content/Images/light-
off.png'</text> } } });
                map.entities.push(pushpin);
                Microsoft.Maps.Events.addHandler(pushpin, 'click', function () {
                    window.location.href = "@Url.Action("DetailsPublicIlluminationCon-
troller", new { Id = location.Id })";
                });
                </text>
            }
        }
    </script>
    <script src="http://www.bing.com/api/maps/mapcontrol?branch=re-
lease&callback=loadMapScenario" async defer></script>
}

<link rel="stylesheet" href="//code.jquery.com/ui/1.11.4/themes/smoothness/jquery-
ui.css">
```

```
<div style="display: none;" id="dialog-confirm" title="Create new Public Illumination
Controller?">
    <p><span class="ui-icon ui-icon-alert" style="float:left; margin:0 7px 20px
0;"></span>Are you sure?</p>
</div>

<h2>Public Illumination Controller</h2>

<p>
    @Html.ActionLink("Create New", "CreatePublicIlluminationController")
</p>

<div id='mapViewer' style='width: 100%; height: 300px;'></div>

<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Latitude)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.Longitude)
        </th>
        <th>
            @Html.DisplayNameFor(model => model.IsOn)
        </th>
        <th></th>
    </tr>

@foreach (var item in Model) {
    <tr>
        <td>
            @item.Latitude.ToString("0.000000")
        </td>
        <td>
            @item.Longitude.ToString("0.000000")
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.IsOn)
        </td>
        <td>
            @Html.ActionLink("Edit", "EditPublicIlluminationController", new
{ id=item.Id }) |
            @Html.ActionLink("Details", "DetailsPublicIlluminationController", new
{ id=item.Id }) |
            @Html.ActionLink("Delete", "DeletePublicIlluminationController", new
{ id=item.Id })
        </td>
    </tr>
}

</table>
```

ANEXO BG – Startup.cs (Site.WebApi)

```
using System;
using Microsoft.Owin;
using Owin;
using Microsoft.Owin.Security.OAuth;
using Site.WebApi.Providers;
```

```
using System.Web.Http;

[assembly: OwinStartup(typeof(Site.WebApi.Startup))]

namespace Site.WebApi
{
    public partial class Startup
    {
        public void Configuration(IAppBuilder app)
        {
            HttpConfiguration config = new HttpConfiguration();

            ConfigureOAuth(app);

            WebApiConfig.Register(config);
            app.UseWebApi(config);
        }

        public void ConfigureOAuth(IAppBuilder app)
        {
            OAuthAuthorizationServerOptions OAuthServerOptions = new OAuthAuthoriza-
tionServerOptions()
            {
                AllowInsecureHttp = true,
                TokenEndpointPath = new PathString("/token"),
                AccessTokenExpireTimeSpan = TimeSpan.FromDays(365 * 100), //token vál-
ido por 100 anos
                Provider = new ApplicationOAuthProvider()
            };

            // Token Generation
            app.UseOAuthAuthorizationServer(OAuthServerOptions);
            app.UseOAuthBearerAuthentication(new OAuthBearerAuthenticationOptions());
        }
    }
}
```

ANEXO BH – ApplicationOAuthProvider.cs

```
using System.Security.Claims;
using System.Threading.Tasks;
using Microsoft.Owin.Security.OAuth;
using Domain.Interfaces.Repositories;
using Data.Tenants.Repositories;
using Domain.Models.Tenants;

namespace Site.WebApi.Providers
{
    public class ApplicationOAuthProvider : OAuthAuthorizationServerProvider
    {
        public override async Task ValidateClientAuthentication(OAuthValidateClientAu-
thenticationContext context)
        {
            context.Validated();
        }

        public override async Task GrantResourceOwnerCredentials(OAuthGrantResource-
OwnerCredentialsContext context)
        {
            ITenantRepository _repo = new TenantRepository();
```

```
            Tenant tenant = _repo.FindByCredentials(context.UserName, context.Pass-
word);

            if (tenant == null)
            {
                context.SetError("invalid_grant", "The user name or password is incor-
rect.");
                return;
            }

            var identity = new ClaimsIdentity(context.Options.AuthenticationType);
            identity.AddClaim(new Claim("sub", context.UserName));
            identity.AddClaim(new Claim("role", "user"));

            context.Validated(identity);
        }
    }
}
```

ANEXO BI – ContainerLevelMeasurerController.cs

```
using Application.Services;
using Application.Services.Interfaces;
using Data.Tenants.Repositories;
using Domain.Interfaces.Repositories;
using Domain.Models.Tenants;
using Domain.Models.Things;
using System.Collections.Generic;
using System.Configuration;
using System.Data.Entity.Infrastructure;
using System.Data.SqlClient;
using System.Linq;
using System.Net;
using System.Security.Claims;
using System.Web.Http;
using System.Web.Http.Description;

namespace Site.WebApi.Controllers
{
    [Authorize]
    public class ContainerLevelMeasurerController : ApiController
    {
        private IContainerLevelMeasurerAppService _clmService;

        private ITenantRepository _tenantRepository;

        private Tenant _currentTenant;

        public ContainerLevelMeasurerController()
        {
            _tenantRepository = new TenantRepository();
        }

        // GET api/thing
        public IEnumerable<ContainerLevelMeasurer> Get()
        {
            IdentityGetter();
            RepositoryGetter();
```

```
            if (_clmService != null)
                return _clmService.GetAll();
            else
                return null;
        }

        // GET api/thing/5
        [ResponseType(typeof(ContainerLevelMeasurer))]
        public IHttpActionResult Get(int id)
        {
            IdentityGetter();
            RepositoryGetter();

            if (_clmService != null)
            {
                ContainerLevelMeasurer thing = _clmService.Find(id);
                if (thing != null)
                {
                    return Ok(thing);
                }
                else
                {
                    return NotFound();
                }
            }
            else
            {
                return BadRequest("Invalid access token");
            }
        }

        // POST api/thing
        [ResponseType(typeof(void))]
        public IHttpActionResult Post(ContainerLevelMeasurer value)
        {
            if (ModelState.IsValid)
            {
                IdentityGetter();
                RepositoryGetter();

                if (_clmService != null)
                {
                    _clmService.Add(value);

                    return CreatedAtRoute("DefaultApi", new { id = value.Id }, value);
                }
                else
                {
                    return BadRequest("Invalid access token");
                }
            }
            else
            {
                return BadRequest(ModelState);
            }
        }

        // PUT api/thing/5
        [ResponseType(typeof(void))]
        public IHttpActionResult Put(int id, ContainerLevelMeasurer value)
        {
            if (!ModelState.IsValid)
```

```
            {
                return BadRequest(ModelState);
            }

            if (id != value.Id)
            {
                return BadRequest();
            }

            IdentityGetter();
            RepositoryGetter();

            if (_clmService != null)
            {
                try
                {
                    _clmService.Update(value);
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!_clmService.GetAll().Contains(value))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
            }
            else
            {
                return BadRequest("Invalid access token");
            }

            return StatusCode(HttpStatusCode.NoContent);
        }

        // DELETE api/thing/5
        [ResponseType(typeof(ContainerLevelMeasurer))]
        public IHttpActionResult Delete(int id)
        {
            IdentityGetter();
            RepositoryGetter();

            if (_clmService != null)
            {
                ContainerLevelMeasurer thing = _clmService.Find(id);
                if (thing == null)
                {
                    return NotFound();
                }

                _clmService.Remove(thing);

                return Ok(thing);
            }
            return BadRequest("Invalid access token");
        }

        #region Helpers
        public void RepositoryGetter()
```

```
        {
            if (_currentTenant.Type.Contains("ContainerLevel"))
            {
                _clmService = ContainerLevelMeasurerAppService.Factory(_cur-
rentTenant.Id, new SqlConnection(ConfigurationManager.ConnectionStrings["IoTData-
base"].ConnectionString), false);
            }
        }

        public void IdentityGetter()
        {
            var userIdentity = (ClaimsIdentity)RequestContext.Principal.Identity;
            var userName = userIdentity.Claims.FirstOrDefault().Value;

            _currentTenant = _tenantRepository.GetAll().Where(t => t.UserName ==
userName).FirstOrDefault();
        }
        #endregion
    }
}
```

ANEXO BJ – PublicIlluminationControllerController.cs

```
using Application.Services;
using Application.Services.Interfaces;
using Data.Tenants.Repositories;
using Domain.Interfaces.Repositories;
using Domain.Models.Tenants;
using Domain.Models.Things;
using System.Collections.Generic;
using System.Configuration;
using System.Data.Entity.Infrastructure;
using System.Data.SqlClient;
using System.Linq;
using System.Net;
using System.Security.Claims;
using System.Web.Http;
using System.Web.Http.Description;

namespace Site.WebApi.Controllers
{
    [Authorize]
    public class PublicIlluminationControllerController : ApiController
    {
        private IPublicIlluminationControllerAppService _picService;

        private ITenantRepository _tenantRepository;

        private Tenant _currentTenant;

        public PublicIlluminationControllerController()
        {
            _tenantRepository = new TenantRepository();
        }

        // GET api/thing
        public IEnumerable<PublicIlluminationController> Get()
        {
            IdentityGetter();
            RepositoryGetter();
```

```
            if (_picService != null)
                return _picService.GetAll();
            else
                return null;
        }

        // GET api/thing/5
        [ResponseType(typeof(PublicIlluminationController))]
        public IHttpActionResult Get(int id)
        {
            IdentityGetter();
            RepositoryGetter();

            if (_picService != null)
            {
                PublicIlluminationController thing = _picService.Find(id);
                if (thing != null)
                {
                    return Ok(thing);
                }
                else
                {
                    return NotFound();
                }
            }
            else
            {
                return BadRequest("Invalid access token");
            }
        }

        // POST api/thing
        [ResponseType(typeof(void))]
        public IHttpActionResult Post(PublicIlluminationController value)
        {
            if (ModelState.IsValid)
            {
                IdentityGetter();
                RepositoryGetter();

                if (_picService != null)
                {
                    _picService.Add(value);

                    return CreatedAtRoute("DefaultApi", new { id = value.Id }, value);
                }
                else
                {
                    return BadRequest("Invalid access token");
                }
            }
            else
            {
                return BadRequest(ModelState);
            }
        }

        // PUT api/thing/5
        [ResponseType(typeof(void))]
        public IHttpActionResult Put(int id, PublicIlluminationController value)
        {
```

```
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            if (id != value.Id)
            {
                return BadRequest();
            }

            IdentityGetter();
            RepositoryGetter();

            if (_picService != null)
            {
                try
                {
                    _picService.Update(value);
                }
                catch (DbUpdateConcurrencyException)
                {
                    if (!_picService.GetAll().Contains(value))
                    {
                        return NotFound();
                    }
                    else
                    {
                        throw;
                    }
                }
            }
            else
            {
                return BadRequest("Invalid access token");
            }

            return StatusCode(HttpStatusCode.NoContent);
        }

        // DELETE api/thing/5
        [ResponseType(typeof(PublicIlluminationController))]
        public IHttpActionResult Delete(int id)
        {
            IdentityGetter();
            RepositoryGetter();

            if (_picService != null)
            {
                PublicIlluminationController thing = _picService.Find(id);
                if (thing == null)
                {
                    return NotFound();
                }

                _picService.Remove(thing);

                return Ok(thing);
            }
            return BadRequest("Invalid access token");
        }

        #region Helpers
```

```
        public void RepositoryGetter()
        {
            if (_currentTenant.Type.Contains("PublicIllumination"))
            {
                _picService = PublicIlluminationControllerAppService.Factory(_cur-
rentTenant.Id, new SqlConnection(ConfigurationManager.ConnectionStrings["IoTData-
base"].ConnectionString), false);
            }
        }

        public void IdentityGetter()
        {
            var userIdentity = (ClaimsIdentity)RequestContext.Principal.Identity;
            var userName = userIdentity.Claims.FirstOrDefault().Value;

            _currentTenant = _tenantRepository.GetAll().Where(t => t.UserName ==
userName).FirstOrDefault();
        }
        #endregion
    }
}
```