Relatório de ARQSOFT – 2ª Parte

1ª Iteração

Apesar desta iteração se focar mais nos CRN, decidimos ter uma iteração mais extensa podendo assim endereçar várias preocupações. As preocupações focadas foram as que bloqueavam avançar o projeto. Para além das preocupações, tivemos atenção às restrições apontadas, sendo realizada uma pequena pesquisa inicial de cada uma, para que quando fossem realizadas as respetivas implementações tivéssemos uma ideia geral do que era necessário fazer.

O grupo teve uma atenção especial em fazer uma análise detalhada aos drivers, podendo assim compreender os vários componentes, as restrições, as preocupações, os casos de uso e os atores.

Etapa 2 - Escolha dos objetivos da iteração

- CRN 1 - Pesquisa sobre multilingua
- CRN 2 - Pesquisa de projetos com arquiteturas semelhantes
- CRN 3 - Pesquisa e documentação de JPA e JDBC
- CRN 4 - Possibilidade de várias pessoas trabalharem em similtânio
- CRN 5 - Modelo de domínio
- CON 1 - Utilização de JPA
- CON 3 - Protótipo acessível através do FrontEnd
- CON 6 - Ferramentas Open Source e Spring
- CON 10 - RestFull Api
- CON 11 - Correr através da linha de comandos o FrontEnd
- CON 12 - Correr através da linha de comandos o BackEnd
- CON 15 - FrontEnd sendo uma SPA
- QA 3 - Configuração das classificações(reviews)
- QA 8 - Utilização de padrões de design

Importância e custo para a implementação e para o negócio

- CRN 1 - Implementação (Médio) Negócio (Médio)
- CRN 2 - Implementação (Médio) Negócio (Médio)
- CRN 3 - Implementação (Médio) Negócio (Baixo)
- CRN 4 - Implementação (Elevado) Negócio (Elevado)
- CRN 5 - Implementação (Elevado) Negócio (Elevado)

- CON 1 - Implementação (Elevado) Negócio (Médio)
- CON 3 - Implementação (Elevado) Negócio (Elevado)

- CON 6 - Implementação (Elevado) Negócio (Elevado)
- CON 10 - Implementação (Elevado) Negócio (Elevado)
- CON 11 - Implementação (Médio) Negócio (Médio)
- CON 12 - Implementação (Médio) Negócio (Médio)
- CON 15 - Implementação (Elevado) Negócio (Médio)
- QA 3 - Implementação (Médio) Negócio (Elevado)
- QA 8 - Implementação (Elevado) Negócio (Elevado)

Etapa 3 - Escolha do que melhorar

Nada a declarar nesta iteração

Etapa 4 - Escolha de conceitos de design que satisfazem os drivers escolhidos

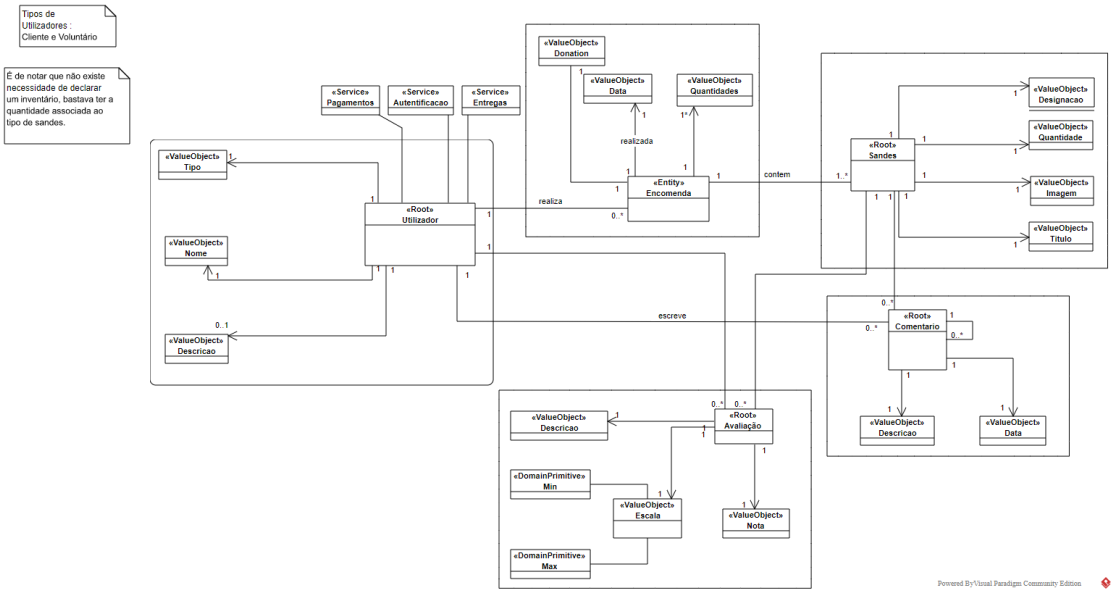
Decisões de design e local de implementação	Racional e suposições
Desenvolver a aplicação de Cliente usando a arquitetura **Web Aplication**	Esta arquitetura é relacionada com o desenvolvimento da aplicação ser acessivel via browser (CON - 3).
Desenvolver a aplicação de Servidor usando a arquitetura **MVC**	O servidor permite ao browser realizar pedidos http, obtendo assim a informação necessária para enriquecer o site. Neste caso foi escolhida uma restFull Api(CON - 10).
Desenvolver o sistema com Spring Boot Framework	O Spring boot foi selecionado devido à restrição descrita no enunciado.

Etapa 5 - Escolha de elementos arquiteturais, alocar responsabilidades e definir interfaces

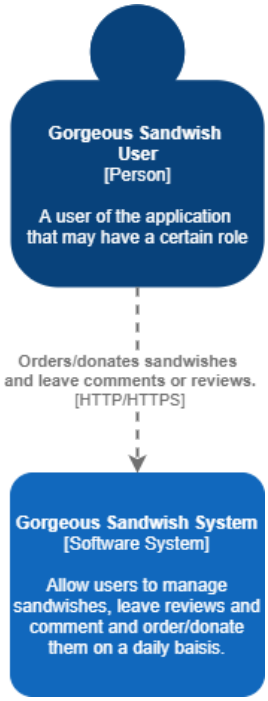
Decisões de design e local de implementação	Racional e suposições
Sistema de traduções simples	Dada a dimensão do projeto a utilização de Json com traduções basta armazenar as traduções num Json File, permitindo assim uma performance otima e sendo bastante configurável pois qualquer alteração necessária é algo simples de realizar.

Etapa 6 - Diagramas e decisões

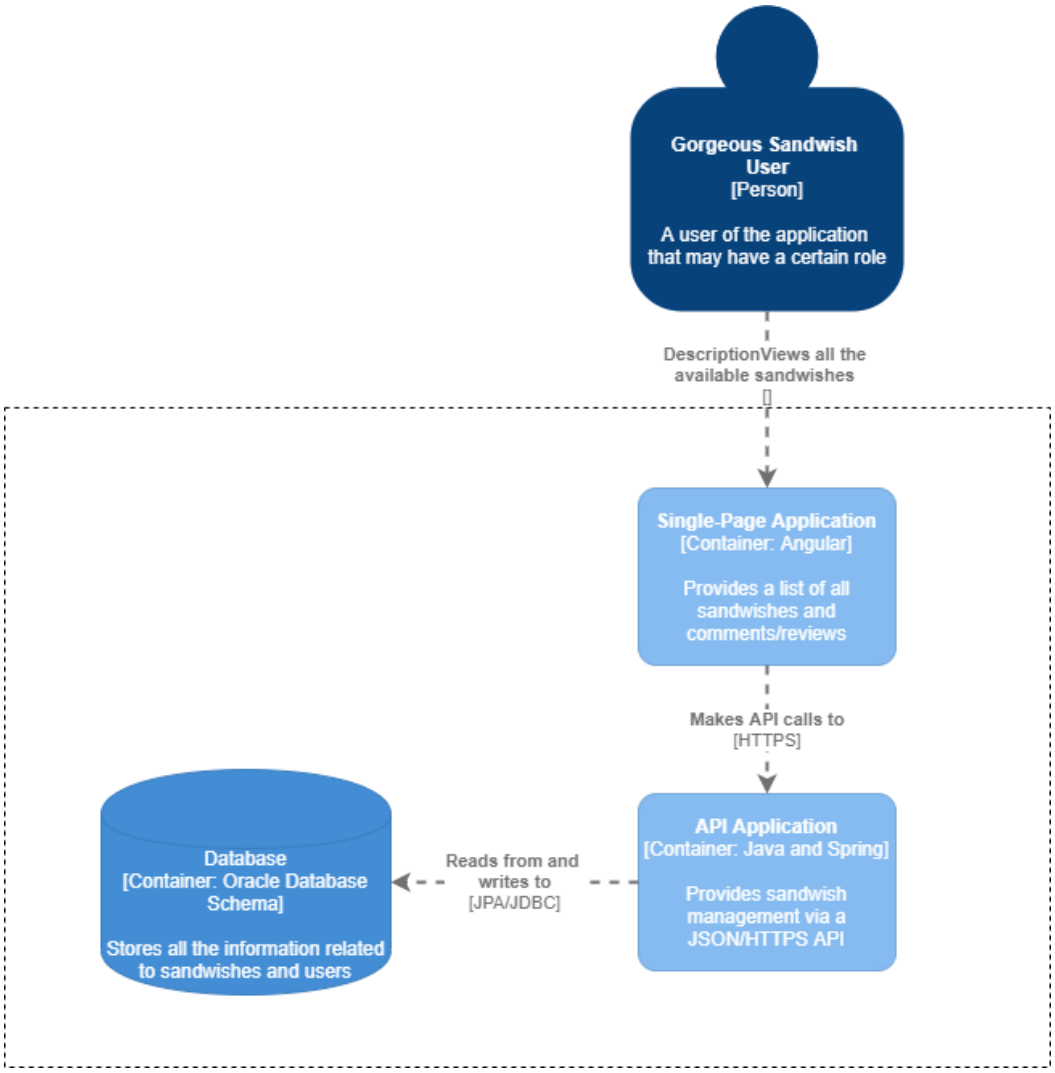
O diagrama seguinte define o modelo de domínio. Para tal despendemos grande parte da iteração para criar algo capaz de implementar todos casos de uso.



Apresentadas as responsabilidades de cada um dos componentes do sistema (Freeman & Freeman, 2011) - Diagrama de nível 1 (System Context diagram)

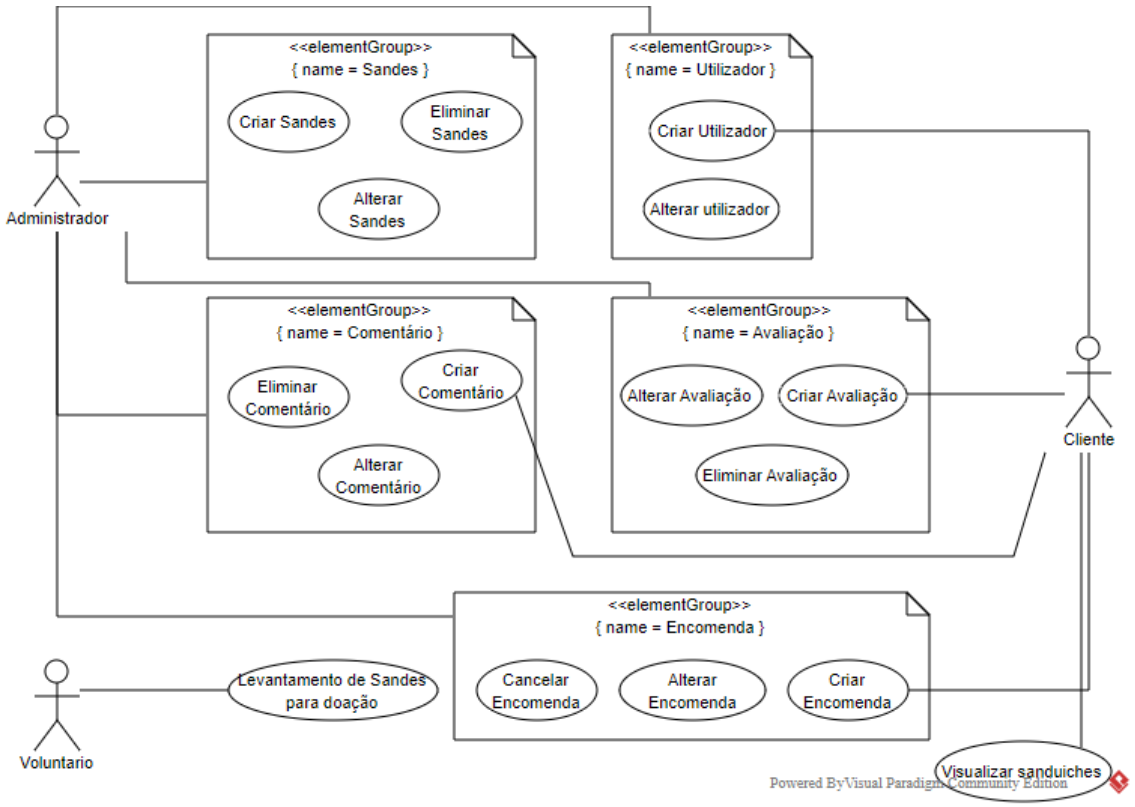


No diagrama apresentado de seguida é possível observar de forma mais detalhada a arquitetura do sistema e como as responsabilidades são divididas - Diagrama de nível 2 (Container diagram)



O utilizador acede à aplicação pelo browser(Edge, Chrome, etc) onde lhe são apresentadas a sandes, os comentários e as avaliações. O FrontEnd realiza pedidos Http/s ao Backend onde é garantido que a transição dos dados é correta. Para além disso realiza as validações e transformações entre o FrontEnd e a Base de Dados, estando a base de dados a persistir a informação.

Atores



Decisões de design e local de implementação	Racional e suposições
Administrador	Capaz de realizar todas as operações do sistema.
Cliente	Utilizador genérico, normalmente apenas tem acesso às funcionalidades "normais".
Voluntário	Apenas realiza a funcionalidade de entrega de sandes para doar.

Etapa 7 - Analise dos objetivos da iteração

Quadro Kanban inicial

1ª Iteração			
Não enderaçado	Endereçado parcialmente	Endereçado completamente	Decisões de design realizadas
CON-1			Identifica a restrição de utilizar JPA devido à equipa já estar habituada e já ter implementação com ela.
CON-11			Identifica a restrição de correr o FrontEnd através da linha de comandos.
CON-3			Identificada a restrição de criar um prototipo acessivel via Web Browser.
CON-6			Identificada a restrição de implementar a RestFull Api com apoio de Spring e a utilização de ferramentas e softwares Open Source. O spring é uma framework que disponibiliza várias atributos que tornaram o projeto mais fiavel e escalavel.
CON-10			Identificada a restrição de criar uma RestFull Api com apoio do Spring para alimentar a UI.
CRN-1			Indentificada a necessidade de uma interface multilingua, neste caso, pelo menos inglês e português (Pesquisas ficaram todas na 1 iteração, para divisão de carga de trabalho).
CRN-2			Necessária pesquisa de projetos com uma arquitetura semelhante para evitar fazer erros que outros já fizeram e seguir dicas/conselhos.
CRN-3			Identificada a necessidade de implementar JPA e JDBC, dada o fraco conhecimento em ambos os temas, foi necessário pesquisar sobre os temas.
CRN-4			Dada a vertente do trabalho identificamos a necessidade de trabalharem varias pessoas em similtário.
CRN-5			Identificada a necessidade de ter um modelo de dominio enriquecido de entidades, objetos, serviços e agregados. Sendo assim foi necessário dispender tempo para alcançar um modelo de dominio válido.
CON-15			Identificada a restrição do FrontEnd/ acesso via browser ser uma Single-Page Aplication
CON-12			Identifica a restrição de correr o BackEnd através da linha de comandos.
QA -7			Identificada a necessidade de adotar padrões de design

CRN-1 -> Relativamente à multilingua, tendo em conta que em princípio será utilizado Angular para aceder ao protótipo através do FrontEnd, foram estudadas 3 opções:

- Objeto JSON com as diferentes traduções para as linguagens desejadas.
- Utilização da API da google - Cloud Transition. (https://cloud.google.com/translate/docs)
- Internationalization (i18n), um mecanismo de adaptação de software a diferentes linguagens. (https://angular.io/guide/i18n)

CRN-2 -> Relativamente à pesquisa de projetos semelhantes concluímos que (https://www.guru99.com/dbms-architecture.html):

- Cada solução deve ser independente das outras, isto é, a alteração de uma não põem em causa a performance das outras.
- Uma solução em 3 camadas permite que cada uma tenha um propósito, potenciando melhores performances.
- Adotando soluções que já existem no mercado facilita certas implementações, por exemplo a configuração de utilização JPA/JDBC (Spring Data Jpa, n.d.).
- Evitar que as diferentes camadas estejam muito agregadas, caso estejam dificultam alterações pois podem comprometer os outros sistemas.
- Concelho 1 - Estar preparado para alterações quer a nível de design ou implementação.
- Concelho 2 - Encontra-se disponível para implementar diferentes tipos de serviços.

CRN-3 -> Relativamente à possibilidade de ter acesso a base de dados, há necessidade de ter o JPA e JDBC configurados.

- A JPA é importante, porque é responsável pela definição de um meio de mapeamento objeto-relacional(ORM) para os objetos.
- A importância do JDBC é fazer a ligação com a base de dados onde envia instruções SQL (29. Working with SQL Databases, n.d.).
- Uma das principais ferramentas de ORM é o Hibernate, sendo atualmente acessível pela especificação do JPA.
- A complexidade de desenvolvimento em JDBC é superior ao JPA, logo o JPA simplifica a manutenção e modificabilidade do programa (Spring Data JDBC Extensions, 2012).

CRN-4 -> Relativamente à possibilidade de várias pessoas a trabalhar no mesmo projeto em simultâneo, é garantido através da utilização de Git.

CRN-5 -> Relativamente ao modelo de domínio foi necessário realizar uma análise detalhada e especifica. Dado o detalhe do relatório e modelos de domínio semelhantes concluímos que:

- É um dos artefactos mais importantes de design, caso não se crie um modelo de domínio capaz de ilustrar os conceitos do problema, torna-se muito mais complexo realizar certas funcionalidades.
- Para o nosso caso é necessário ter um modelo escalável.
- Para alcançar mais detalhe, decidimos utilizar agregados, entidades, raízes e serviços.

Dada a complexidade e dimensão do projeto a desenvolver (apenas duas linguagens necessárias e pouca informação a ser disponibilizada no FrontEnd), optou-se por utilizar a primeira opção. A API da google seria ideal para permitir traduções para um número elevado de linguagens e, para além disso, a consistência entre as diferentes linguagens tornar-se-ia inconsistente. Relativamente ao i18n, também seria algo recomendado para um projeto que necessitasse de um número maior de linguagens.

QA 3 -> Relativamente à possibilidade ser possível ter as classificações configuráveis.

- Utilizada uma ficheiro de configurações, ao iniciar as classificações, é obtido do ficheiro os valores parametrizados.

QA 8 -> Relativamente à utilização de padrões de design

- Foram estudados vários padrões a utilizar, desde a boas práticas de design às de codificações.
- Utilizado Single Responsability Principle.
- Criar o projeto com o padrão repository.

CON-15 -> Tendo em conta que o acesso pelo web browser deve ser feito através de uma SPA, foi decidido utilizar Angular devido à experiência da equipa de desenvolvimento em utilizar a framework. (https://angular.io/)

Quadro Kanban final

2ª Iteração

A escolha dos objetivos desta iteração baseiam-se no inicio da implementação dos casos de uso, criando um suporte primário capaz de realizar os casos de uso mais relevantes. Após a sua implementação é mais fácil detalhar as suas regras de negócio.

Os CRN em falta também foram considerados nesta iteração e os drivers que não foram completados na iteração anterior.

Etapa 2 - Escolha dos objetivos da iteração

- UC 1 - Criar sanduiche
- UC 2 - Eliminar sanduiche
- UC 3- Alterar sanduiche
- UC 4 - Criar comentário
- UC 5 - Eliminar comentário
- UC 6 - Criar avaliação
- UC 7 - Eliminar avaliação
- UC 9 - Criar Encomenda
- UC 10 - Alterar Encomenda
- CRN 6 - Autenticação
- CRN 7 - Autorização
- CON 2 - Utilização de JDBC
- CON 4 - Encomenda da sandes entre 3 a 28 dias
- CON 5 - Alteração da encomenda até 5 dias
- CON 9 - Implementação de multilingua
- CON 13 - Sistema compatível com sistema de entregas e pagamentos
- QA 3 - Configuração de classificações
- QA 6 - Utilização de DTO's
- QA 7 - Utilização do padrão Builder
- QA 8 - Utilização de padrões de design

Importância e custo para a implementação e para o negócio

- UC 1 - Implementação (Médio) Negócio (Elevado)
- UC 2 - Implementação (Médio) Negócio (Médio)
- UC 3 - Implementação (Médio) Negócio (Elevado)
- UC 4 - Implementação (Médio) Negócio (Elevado)
- UC 5 - Implementação (Médio) Negócio (Médio)
- UC 6 - Implementação (Médio) Negócio (Elevado)
- UC 7 - Implementação (Médio) Negócio (Médio)
- UC 9 - Implementação (Elevado) Negócio (Elevado)

- UC 10 - Implementação (Elevado) Negócio (Elevado)
- CRN 6 - Implementação (Baixo) Negócio (Elevado)
- CRN 7 - Implementação (Baixo) Negócio (Elevado)
- CON 2 - Implementação (Elevado) Negócio (Elevado)
- CON 4 - Implementação (Médio) Negócio (Elevado)
- CON 5 - Implementação (Médio) Negócio (Médio)
- CON 9 - Implementação (Médio) Negócio (Elevado)
- CON 13 - Implementação (Médio) Negócio (Médio)
- QA 3 - Implementação (Médio) Negócio (Elevado)
- QA 6 - Implementação (Médio) Negócio (Médio)
- QA 7 - Implementação (Médio) Negócio (Baixo)
- QA 8 - Implementação (Médio) Negócio (Alto)

Etapa 3 - Escolha do que melhorar

O que foi decidido para melhorar foram os módulos da iteração anterior em atraso e o módulo de domínio.

Etapa 4 - Escolha de conceitos de design que satisfazem os drivers escolhidos

Decisões de design e local de implementação	Racional e suposições
Criar um domínio para a aplicação	Antes de se começar a implementar os casos de uso é necessário criar um modelo capaz de satisfazer as necessidades do negócio. Para tal são criadas a entidades e as suas relações.
Identificar os objetos capazes de mapear as funcionalidades	Tendo cada objeto um propósito específico, este é responsável por determinadas funcionalidades, sendo assim promovidas responsabilidades únicas.
Uso de JPA	Permite a utilização de ORM's para mapeamento de objetos, é bastante usada em paralelo com spring. Outras ferramentas não foram consideradas pelo facto de esta ser usada.

Etapa 5 - Escolha de elementos arquiteturais, alocar responsabilidades e definir interfaces

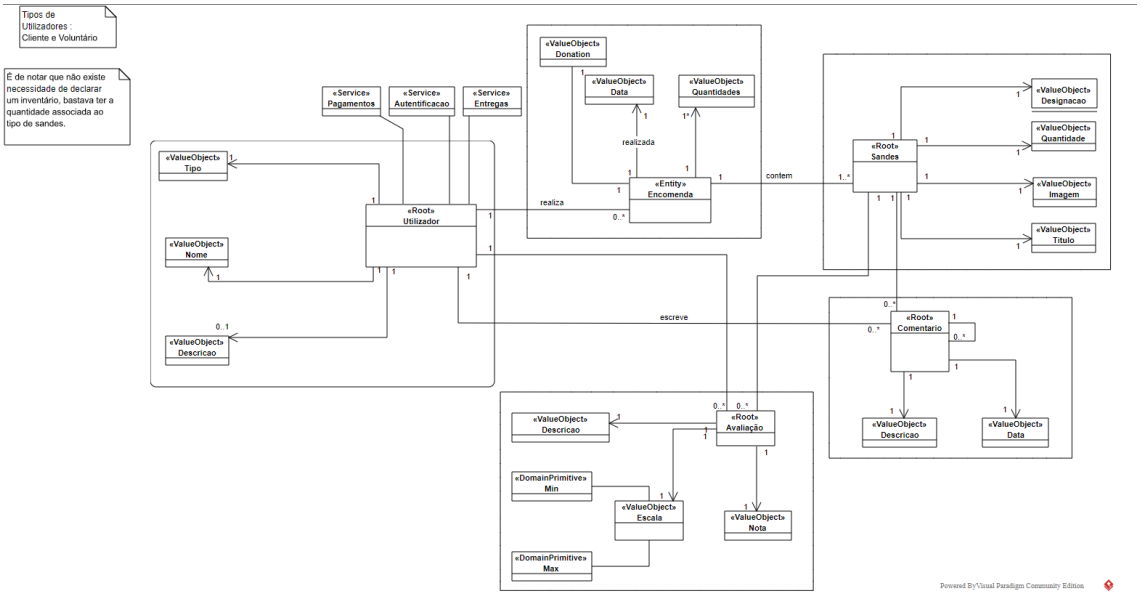
Decisões de design e local de implementação	Racional e suposições
Conexão entre módulos com apoio de Spring	A gestão das dependências foi facilitada com apoio da Spring, por exemplo a injeção de componentes é facilitada com o "Autowired".
Mapeamento dos	A anotação permite mapear os objetos com grande flexibilidade

objetos	e sem grande esforço.
Refatorização do modelo de domínio	Com a implementação dos casos de uso foi percetível que era necessário realizar certos ajuste.
Criar um sistema aberto a alterações	É percetível que uma aplicação deste tipo tem de estar disponível a expansão, sendo possível adicionar funcionalidades externas sem comprometer a sua estabilidade (CON 13).
Regras de negócio configuráveis	A implementação de regras de negócio como validações (CON 4, CON 5) estão abertas a qualquer mudança onde o funcionamento da aplicação não é comprometido.

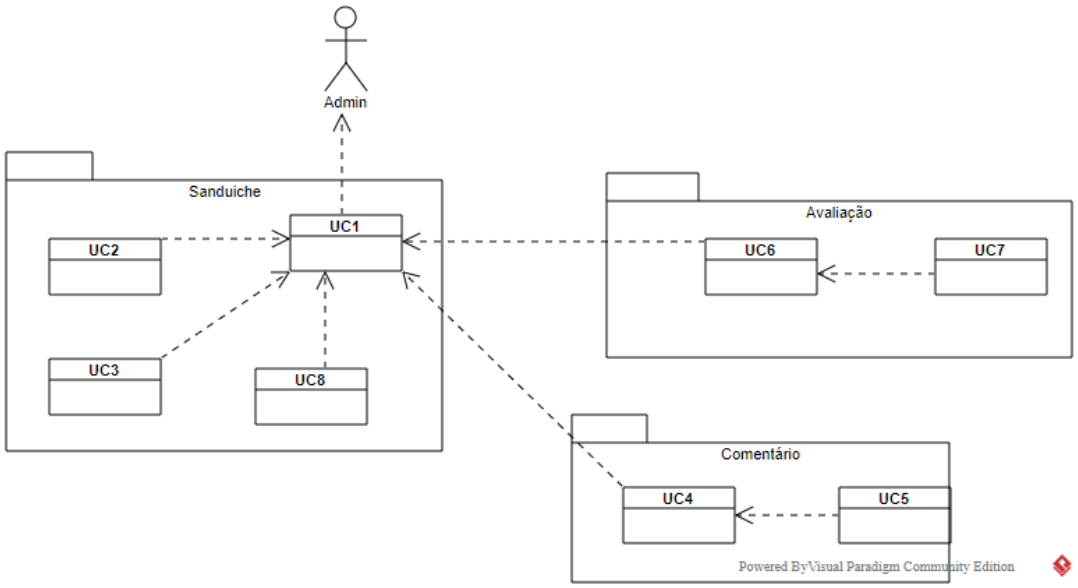
Etapa 6 - Diagramas e decisões

Decisões de design e local de implementação	Racional e suposições
Ator	Responsável por realizar ações sobre os sistemas.
Presentation	Este módulo contêm módulos que controlam a interação do utilizador e o seu fluxo.
Controller	Esta camada é responsável por controlar o fluxo de informação entre back e front end.
Service (Logica de negócio)	Contêm as operações que mantêm a logica de negócio.
Database	Persistência de dados.
Model	Modela os os domínios existentes no negócio.
Utils	Utilizado entre vários módulos, permite os métodos chamarem outros métodos de forma a reduzir linhas de código e facilitar a codificação.
DTO	Este módulo garante que informação confidencial não é extraída de certos módulos, garantido segurança dos dados.

O diagrama seguinte define o modelo de domínio, este foi refactorizado nesta iteração.

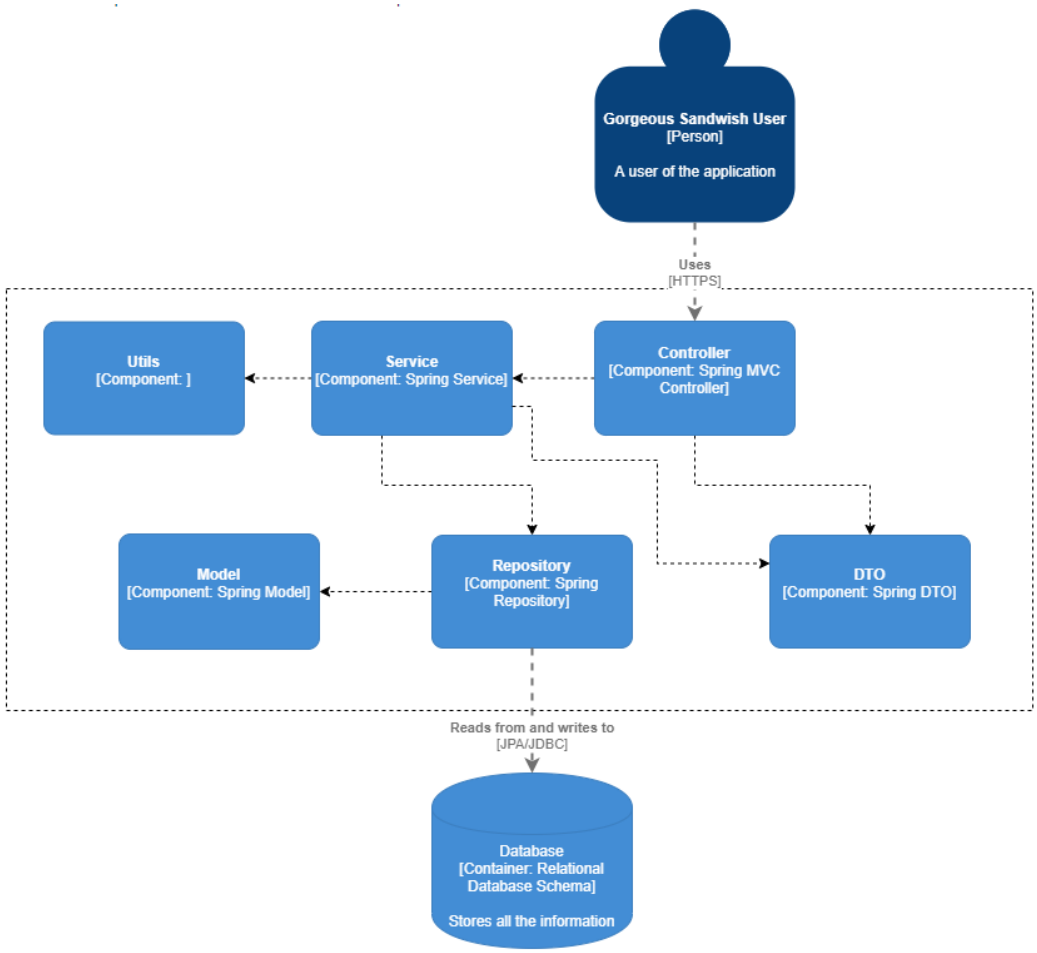


Representação das responsabilidades entre Casos de Uso.

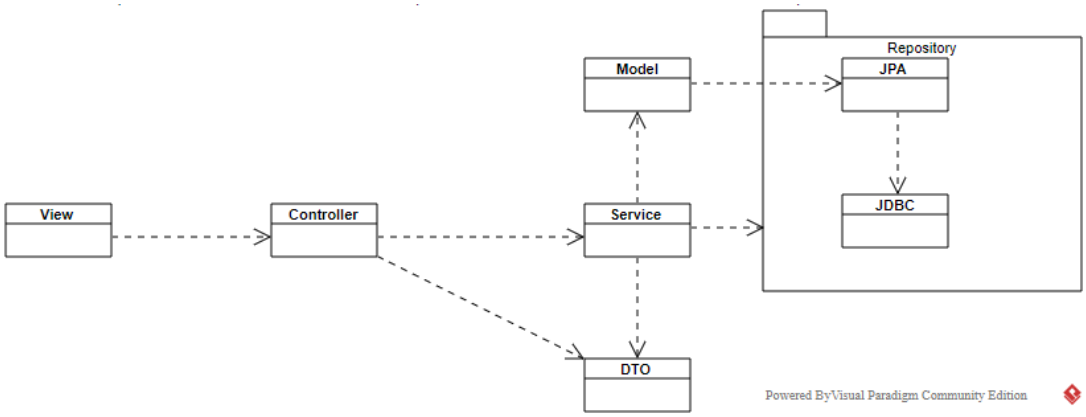


Apresentadas as responsabilidades de cada um dos componentes da solução

- Detalha as responsabilidades de cada um dos componentes (Guru99, 2019).

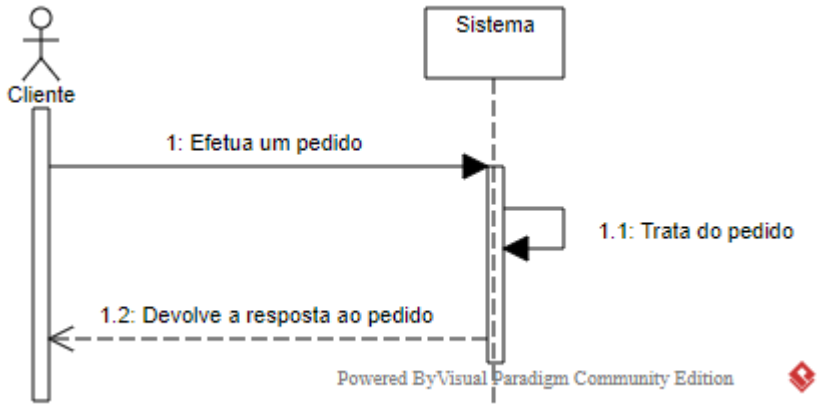


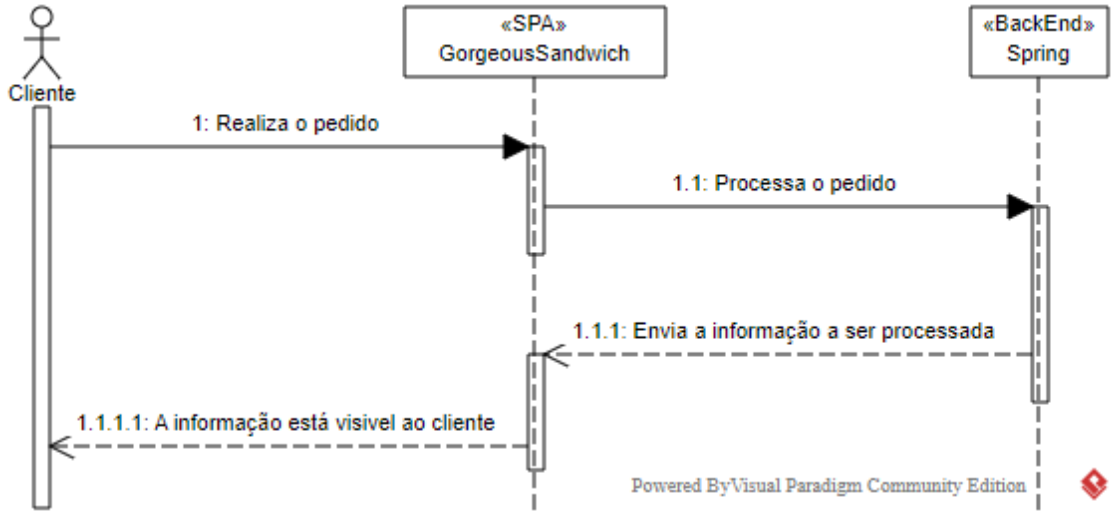
- Detalha as responsabilidades de cada um dos componentes, entrando em detalhe na camada de persistência. Responsabilidades Internas



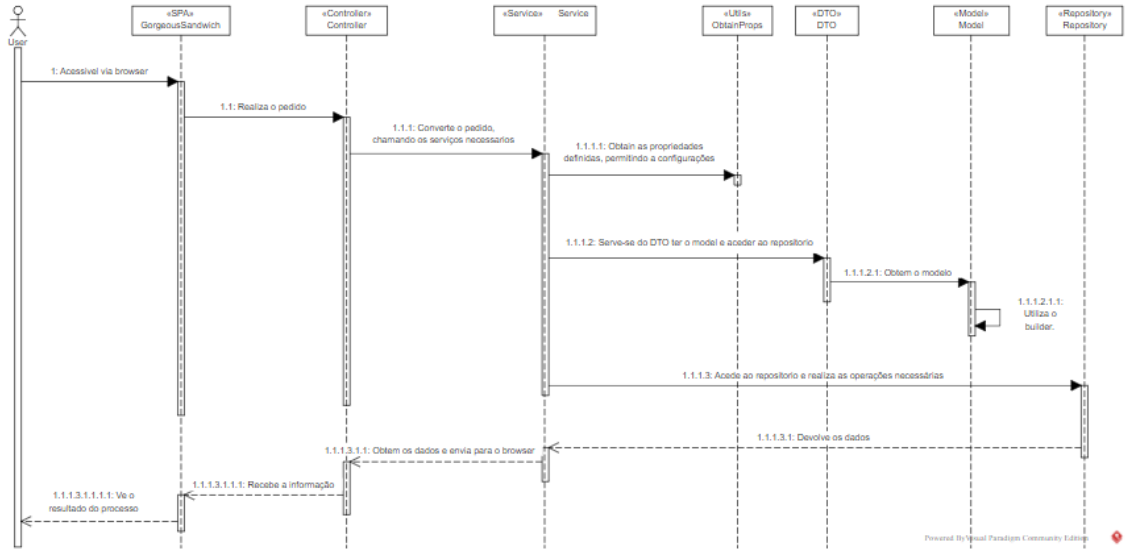
O diagrama de sequência apresentado é genérico, demonstrando o fluxo da informação de forma a entender o que realiza em cada um dos módulos. Foram criados mais diagramas, mas não sendo necessário entrar em detalho decidimos não os colocar no relatório.

- O Cliente realiza o pedido, o sistema recebe a informação do pedido, trata-a e devolve-a.





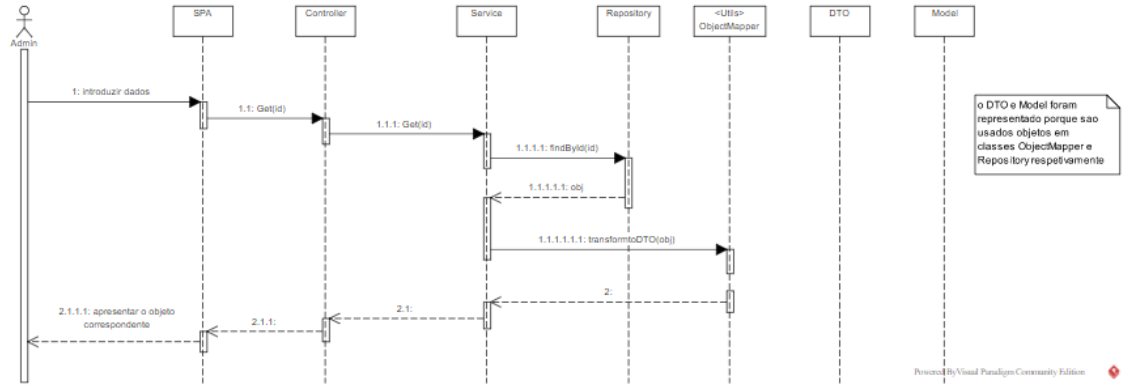
- O Cliente acede ao serviço pelo browser, pela SPA, esta realiza a comunicação com o backend, enviando um pedido, o backend recebe o pedido, processa a informação, devolvendo ao cliente.

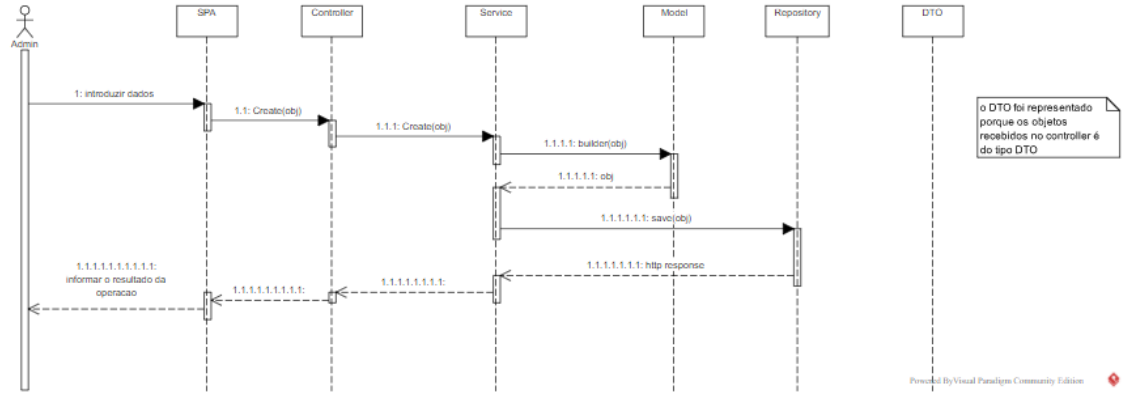


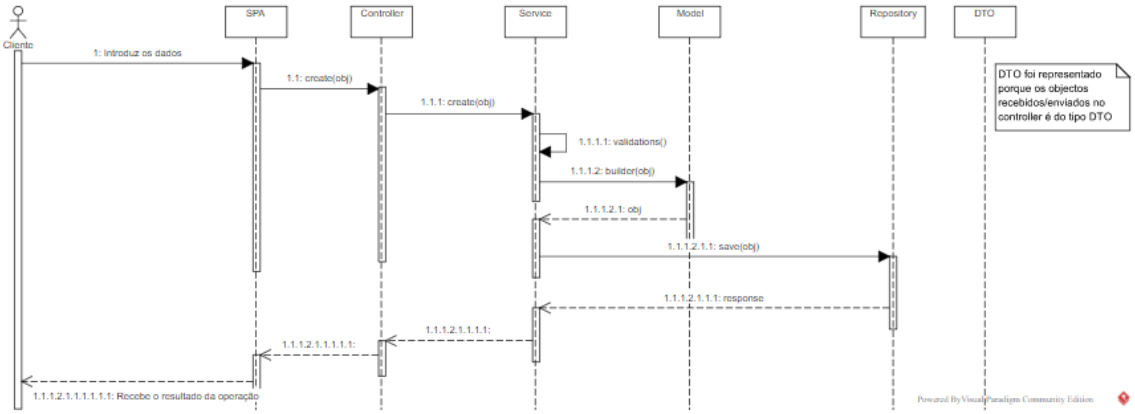
Explicação detalhado do fluxo

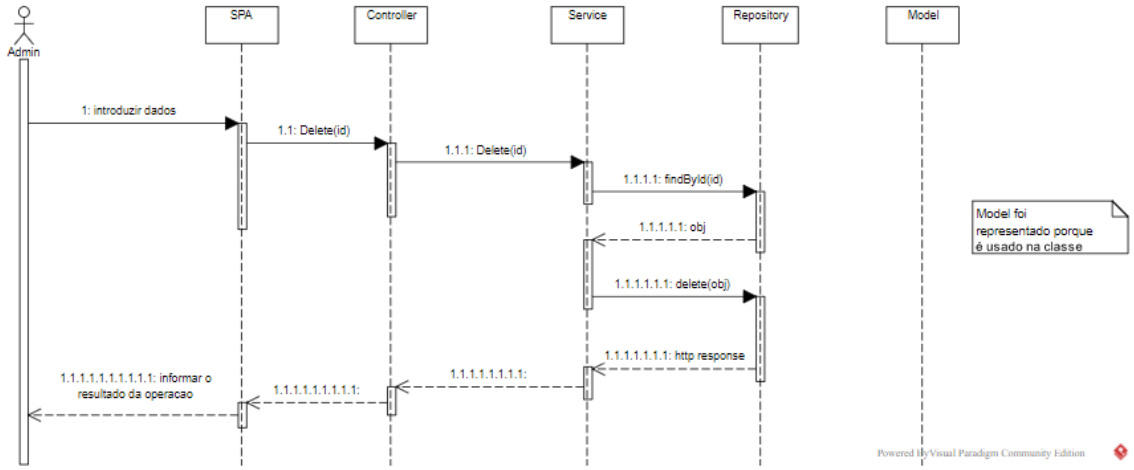
- O utilizador introduz o link para a GorgeousSandwich, tendo lá um site que permite mudar a linguagem de apresentação e onde são apresentadas sandes com os seus comentários e avaliações. Estas informações vêm do backend.
- O Controller recebe o pedido das informações necessárias, conforme a parametrização dos dados é interpretada qual deve a informação a obter.
- O Serice restrutura a informação e realiza o pedido ao repositório, cria ou decompõe os objetos e obtém propriedades fulcrais para configurar o sistema.
- O Dto garante que no fluxo de informação são entrem nem sai informação confidencial.
- O Model cria o objeto capaz de realizar as funcionalidades de negócio.
- O Repositório garante que os dados guardados estão corretos, permitindo realizar pesquisas a estes mesmos.

Para além dos diagramas genéricos o grupo decidiu realizar um diagrama de CRUD.









Divisão de responsabilidades

De forma a entender melhor os métodos criados e as responsabilidades associadas, são de seguida explicados os mesmos visto que o desenho da solução nesta iteração envolveu a sua implementação. Neste caso, visto que foi realizado um CRUD básico para todos os modelos existente, serão inicialmente explicados de um modo geral os métodos associados e depois será feita uma análise de casos mais específicos relacionados com alguns dos modelos tendo em conta as restrições, preocupações e atributos de qualidade.

Repository:

Método	Descrição
save(Model model)	Responsável por persistir uma nova entidade na base de dados.
findAll()	Responsável por retornar a lista de todas as entidades existentes na base de dados.
findById(long id)	Responsável por retornar uma entidade com um determinado id da base de dados.
delete(Model model)	Responsável por apagar uma determinada entidade da base de dados através do id.

Service:

Esta componente encontra-se diretamente ligada à falada anteriormente através dos métodos referidos, sobre a qual processa todas as validações necessárias.

Método	Descrição
createModel(ModelDTO modelDTO)	Responsável por validar o objeto e a persistir o mesmo caso passe nas validações.
updateModel(long id, ModelDTO modelDTO)	Responsável por validar os dados a alterar e através do id persistir os novos campos dessa entidade.
getAllModels()	Responsável por retornar todas as entidades no formato desejado (DTO).
getModelById(long id)	Responsável por retornar uma determinada entidade através do id no formato desejado (DTO).
deleteModel(long id)	Responsável por apagar uma determinada entidade através do id.

É também importante referir a existência de método diferente no modelo das **Sandes**:

Método	Descrição
getAllSandesDetailed()	Responsável por retornar todas as sandes existentes no sistema e os comentário e avaliações associados.

Controller:

Esta componente encontra-se ligada à anterior e é responsável por decidir como o utilizador interaje com a aplicação.

Método	Descrição
createModel(ModelDTO modelDTO)	Responsável por criar um novo modelo através dos dados inseridos pelo utilizador.
updateModel(long id, ModelDTO modelDTO)	Responsável por alterar um modelo tendo em conta os dados inseridos pelo utilizador.
getAllModels()	Responsável por listar todos os modelos e retorná-los para a rota associada.
getModelById(long id)	Responsável por apresentar ao utilizador um modelo específico através do seu id.
deleteModel(long id)	Responsável por apagar um modelo a partir do seu id.

Assim como nos serviços, o controller das **Sandes** também contém o método mais importante de toda a aplicação. Este será o endpoint que irá fazer comunicação com o fronted:

Método	Descrição
getAllSandesDetailed()	Responsável por listar ao utilizador todas as sandes existentes no sistema e os comentário e avaliações associados às mesmas.

Utils:

Nesta componente encontram-se implementados todos os métodos adicionais, responsáveis pela lógica do projeto, os quais são utilizados por todas as outras componentes.

Método	Descrição
isBetweenDates(Date date)	Responsável por verificar se uma dada data se encontra entre um determinado período, definido por um ficheiro de configuração.
getDateFromDaysApart(int daysApart)	Responsável por retornar a data atual mais os dias inseridos na variável "daysApart".
isHourBetweenInterval(String horaRequisitada)	Responsável por verificar se um hora se encontra entre duas horas predefinidas por um ficheiro de configuração.
map(Model model, ModelDTO dto)	Responsável por retornar o DTO respetivo de um dado modelo.
mapAll(ModelList modelList, DTOList dtoList)	Responsável por retornar uma lista de DTOs respetiva de um dada lista de modelos.

getPropertiesValue(String key)	Responsável por retornar uma determinada propriedade tendo em conta a chave.

Etapa 7 - Analise dos objetivos da iteração

Quadro Kanban inicial

CRN-6 -> Relativamente à autenticação, tendo em conta a necessidade de análise, mas não de implementação:

- O frontEnd deve estar disponível à eventualidade de ser necessário realizar "login".
- O backend deve ser capaz de processar a autenticação, garantindo que os utilizadores apenas realizam funções que têm permissões.
- Deve ser possível alterar o módulo do utilizador de forma a garantir os pontos acima descritos.

CRN 7 - Relativamente à autorização, apenas é necessário espelhar:

- Os utilizadores apenas têm acesso ao que é permitido.

- Ao criar um utilizador, caso não seja indicado, é lhes atribuído o mínimo de permissões.

CON 4 - Relativamente à restrição da encomenda da sandes ser entre 3 a 28 dias:

- Ao criar a encomenda é validade se a data da encomenda se encontra dentro da gama de dias definido.

CON 5 - Relativamente à restrição de alteração da encomenda até 5 dias:

- Ao alterar a encomenda é validade se a data entrega da encomenda é superior ou igual a 5 dias.

QA 6 - Utilização de DTO's

- Criar DTO's para a transferência de dados entre camadas, assim garantindo a confidencialidade dos dados.

QA 7 - Utilização do padrão Builder

- Foi criado para cada classe model um builder para ajudar na construção segura dos objetos favorecendo a integridade dos dados.

Quadro Kanban final

3ª Iteração

A escolha dos objetivos da iteração consiste em verificar o que falta e o que melhorar e desenvolver.

Etapa 2 - Escolha dos objetivos da iteração

- UC 8 - Registo de levantamento de Sandes para doação
- UC 11 - Visualizar dados
- CON 7 - Configuração do ultimo intervalo de entrega
- CON 8 - Escrita da avaliação da sande restrita a quem experimentou
- CON 14 - Implementação em Docker
- QA 1 - Configuração de usar JPA/JDBC
- QA 2 - Configuração de intervalos de entrega e horas de entrega
- QA 4 - Testar FrontEnd
- QA 5 - Testar BackEnd e serviços

Importância e custo para a implementação e para o negócio

- UC 8 - Implementação (Médio) Negócio (Elevado)
- UC 11 - Implementação (Elevado) Negócio (Elevado)
- CON 7 - Implementação (Elevado) Negócio (Elevado)
- CON 8 - Implementação (Elevado) Negócio (Elevado)
- CON 14 - Implementação (Elevado) Negócio (Elevado)
- QA 1 - Implementação (Elevado) Negócio (Médio)
- QA 2 - Implementação (Elevado) Negócio (Médio)
- QA 4 - Implementação (Baixo) Negócio (Elevado)
- QA 5 - Implementação (Baixo) Negócio (Elevado)

Etapa 3 - Escolha do que melhorar

O que foi decidido para melhorar foi módulos da iteração anterior em atraso e o módulo de domínio.

Etapa 4 - Escolha de conceitos de design que satisfazem os drivers escolhidos

Decisões de design e local de implementação	Racionais e suposições
Usar docker	Docker é uma plataforma que permite a virtualização da entrega de software(CON 14), criando contentores. Contentores estão isolados entre si. Docker é gratuito e dispõe de diversas funcionalidades adicionais.
Testes FrontEnd	Permite verificar o comportamento dos vários componentes do GorgeousSandwich (QA -4). Neste caso o grupo decidiu adotar teste

	de comportamento, verificando visualmente se a funcionamento da aplicação estava correto.
Testes BackEnd	Permite validar diversos comportamentos da RestFull Api (QA - 5), foram usados diversos testes neste caso. Testes unitários aos modelos, testes de comportamento e testes onde validamos se o fluxo da informação é correto (ex: Ao criar, obter, alterar, obter e eliminar, seguindo esta ordem garantimos a atomicidade da informação e que os métodos garantem segurança nos dados).

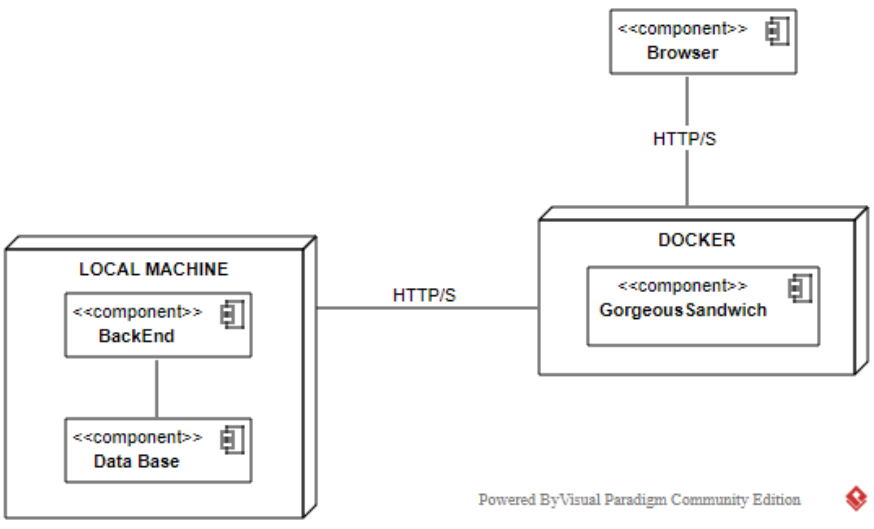
Etapa 5 - Escolha de elementos arquiteturais, alocar responsabilidades e definir interfaces

Decisões de design e local de implementação	Racionais e suposições
Conexão entre módulos com apoio de Spring	A gestão das dependências foi facilitada com apoio do Spring, por exemplo, a injeção de componentes é facilitada com o "Autowired".
Mapeamento dos objetos	A anotação permite mapear os objetos com grande flexibilidade e sem grande esforço.
Refatorização do modelo de domínio	Com a implementação dos casos de uso foi percetível que era necessário realizar certos ajustes.
Criar um sistema aberto as alterações	É percetível que uma aplicação deste tipo tem de estar disponível a expansão, sendo possível adicionar funcionalidades externas sem comprometer a sua estabilidade(CON 13).
Regras de negócio configuráveis	A implementação de regras de negócio como validações (CON 4,CON 5) estão abertas a mudança onde qualquer não compromete o funcionamento da aplicação.

Etapa 6 - Diagramas e decisões

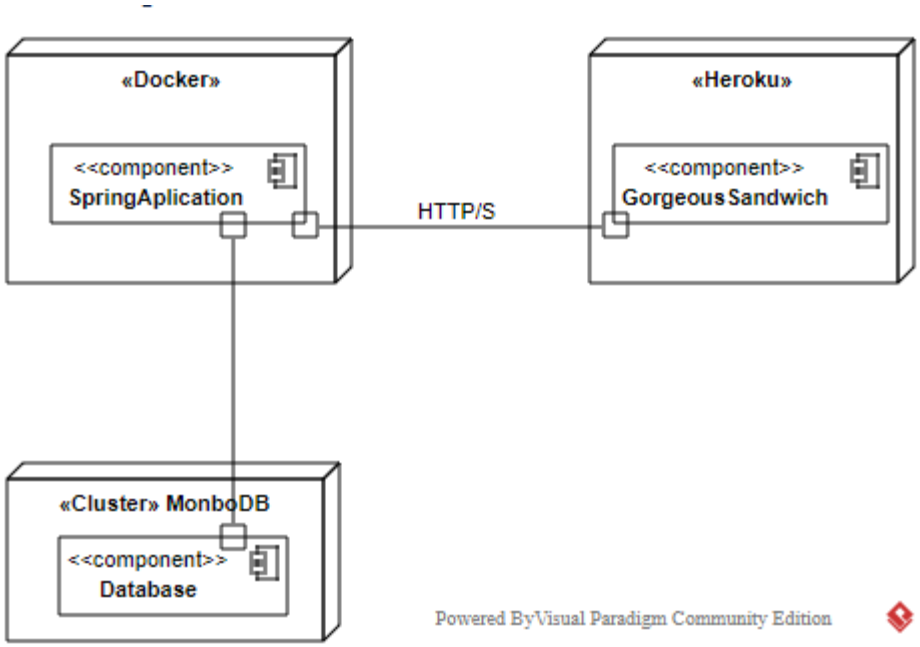
O diagrama seguinte consiste num diagrama de Deployment.

- Este diagrama consiste no diagrama implementado.
- Por questões de ser mais simples, melhorar a performance e cumprir os requisitos, o backend e a base de dados encontram-se na máquina e o frontend corre num



contentor Docker.

- Esta alternativa demonstra a capacidade de colocar cada sistema como independente, o frontend em cloud no **Heroku** continuando a comunicar com o backend através de pedidos (HTTP/S), o backend no Docker e a base de dados num **Cluster** de MongoDb.



Etapa 7 - Analise dos objetivos da iteração

ATAM

Para a avaliação do ATAM em relação ao risco, foi levado em consideração o impacto que um determinado atributo de qualidade teria no sistema caso o mesmo estiver indisponível. Na classificação o 3, 6 e o 9 significam baixo, médio e alto respetivamente. Foi identificado o tipo do atributo e a razão pela qual existem.

Cenário QA-1	**Protótipo compatível com JPA e JDBC**
Atributo	Modificabilidade e portabilidade
Razão	Poder modelar os dados e aceder a Base de dados
Risco	9
Cenário QA-2	**Configuração de intervalos de entrega e horas de entrega**
Atributo	Modificabilidade
Razão	Capacidade de alterar os intervalos de entrega e horas<p> de entrega através de um ficheiro de configuração
Risco	1
Cenário QA-3	**Configuração dos intervalos das classificações das sanduíches**
Atributo	Modificabilidade
Razão	Capacidade de alterar os limites das classificações<p> através de um ficheiro de configuração
Risco	3
Cenário QA-4	**Testar FrontEnd**
Atributo	Testabilidade
Razão	Garantir que os dados apresentados estão corretos
Risco	8

Cenário QA-5	**Testar BackEnd e serviços**
Atributo	Testabilidade
Razão	Garantir que as funcionalidades funcionam corretamente
Risco	8

Cenário QA-6	**Utilização de DTO's**
Atributo	Confidencialidade e integridade
Razão	Não são disponibilizados dados que podem comprometer a<p> segurança dos dados, como por exemplo, o id dos objetos
Risco	6

Cenário QA-7	**Utilização do padrão Builder**
Atributo	Integridade
Razão	Garantir que todos os dados criados estão certos
Risco	6

Cenário QA-8	**Utilização de padrões de design**
Atributo	Restrição de Implementação
Razão	Obedecer uma estrutura de desenvolvimento do código
Risco	5

Quadro Kanban inicial

CON - 7 -> Registo de levantamento de Sandes para doação:

- Só é permitido o levantamento de uma sande para a doação quando o utilizador é um voluntário.

CON -8 -> Escrita da avaliação da sande restrita a quem experimentou:

- Criada uma restrição nos serviços que permite que apenas quem encomendou uma sande pode fazer uma avalição do mesmo.

CON -14 -> Relativamente à implementação em Docker:

- Foi detetado que devia ser implementado em Docker.

- Por questões de performance e prova de conceito, decidimos implementar apenas uma das aplicações, demonstrando assim que é possível comunicação entre as aplicações, mesmo que estas estejam em contentores diferentes.
- Para implementar aplicações no Docker, é necessário compilar. Caso dese imagens de aplicações java é necessário fazer **Build** criando um **Jar**, caso seja o frontEnd é criado **Package Dist** que contêm os ficheiros compilados. `ng build --prod`
- É criado de seguida uma imagem que contêm a aplicação, o detalhe da criação da imagem depende do **DockerFile**.

```
PS C:\Users\morei\Documents\ISEP\Mestrado\ARQSOFT\Git-ARQSOFT-Grupo\part 2\projects\frontend\gorgeous-sandwich> docker build -t
gorgeoussandwich:v2 .
[+] Building 2.7s (7/7) FINISHED
 => [internal] load .dockerignore                                                                                    0.1s
 => => transferring context: 2B                                                                                      0.1s
 => [internal] load build definition from Dockerfile                                                                 0.1s
 => => transferring dockerfile: 113B                                                                                 0.1s
 => [internal] load metadata for docker.io/library/nginx:1.17.1-alpine                                               2.3s
 => [internal] load build context                                                                                    0.2s
 => => transferring context: 489.95kB                                                                                0.1s
 => CACHED [1/2] FROM docker.io/library/nginx:1.17.1-alpine@sha256:17bd1698318e9c0f9ba2c5ed49f53d690684dab7fe3e8019b855c3  0.0s
 => [2/2] COPY /dist/gorgeous-sandwich /usr/share/nginx/html                                                         0.1s
 => exporting to image                                                                                               0.1s
 => => exporting layers                                                                                              0.0s
 => => writing image sha256:441907088e7d40ffded63a03816586fc0700b3a33a1bdaab4c1a07b748812379                         0.0s
 => => naming to docker.io/library/gorgeoussandwich:v2                                                               0.0s
```

```
PS C:\Users\morei\Documents\ISEP\Mestrado\ARQSOFT\Git-ARQSOFT-Grupo\part 2\projects\frontend\gorgeous-sandwich> docker run -p 80
:80 gorgeoussandwich:v2
```

- **QA 1 -> Configuração de usar JPA/JDBC:**
- Usou o JPA para poder modelar os dados colocando as anotações nas classes model.
- O JDBC foi usado para fazer a ligação com a Base de dados.
- **QA 2 -> Configuração de intervalos de entrega e horas de entrega:**
- Capacidade de alterar os intervalos de entrega e horas de entrega através de um ficheiro de configuração.
- **QA 4 -> Testar FrontEnd:**
- Foram feitos testes de usabilidades para garantir que as funcionalidades estão certos.
- **QA 5 -> Testar BackEnd e serviços:**
- Foram feitos testes unitários para garantir a consistência dos dados.

Quadro Kanban final

Referências

29. Working with SQL databases. (n.d.). Retrieved November 18, 2020, from https://docs.spring.io/spring-boot/docs/1.4.1.RELEASE/reference/html/boot-features-sql.html

Freeman, A., & Freeman, A. (2011). Authentication and Authorization. In *Applied ASP .NET 4 in Context* (pp. 847–874). https://doi.org/10.1007/978-1-4302-3468-5_34

Guru99. (2019). *DBMS Architecture: 1-Tier, 2-Tier & 3-Tier*. https://www.guru99.com/dbms-architecture.html

Spring Data JDBC Extensions. (2012). https://spring.io/projects/spring-data-jdbc

spring data jpa. (n.d.). Retrieved November 18, 2020, from https://spring.io/projects/spring-data-jpa