

TornGes Software Architecture Documentation

FC47806 - Paulo Santos
FC47878 - Rafael Melo
FC47888 - Diogo Pereira

25 de Novembro de 2018

Conteúdo

1 Visão Geral e Mapeamento da Documentação	4
1.1 Propósito e <i>Scope</i> do SAD	4
1.2 Organização do SAD	4
1.3 Representação dos <i>Stakeholders</i>	5
1.4 Como uma vista é documentada	5
2 Background da Arquitetura	6
2.1 Descrição do Problema	6
2.1.1 Contexto	6
2.1.1.1 Contexto Técnico	6
2.1.1.2 Contexto do Ciclo de Vida do Projeto	6
2.1.2 Requisitos Funcionais	6
2.1.2.1 Caso de Uso 1 - Criar Torneio	7
2.1.2.2 Caso de Uso 2 - Registar Participante em Torneio	8
2.1.2.3 Caso de Uso 3 - Registar Resultado de Encontro	9
2.1.2.4 Caso de Uso 4 - Visualizar Calendário de Jogador	10
2.1.3 Requisitos de Qualidade	11
2.1.3.1 Desempenho	11
2.1.3.2 Fiabilidade:	11
2.1.3.3 Modificabilidade	11
2.1.3.4 Usabilidade	12
2.1.3.5 Testabilidade	12
2.1.3.6 Interoperabilidade	12
2.1.3.7 Potencial de Reutilização:	12
2.2 <i>Background</i> da Solução	14
2.2.1 Solução Arquitetural	14
3 Vistas Arquiteturais	15
3.1 Vistas	15
3.1.1 Vistas de Módulos	15
3.1.1.1 Vista de Decomposição do <i>Business</i>	15
3.1.1.2 Vista de Decomposição do <i>Web</i>	18
3.1.1.3 Vista de Decomposição do <i>GUI</i>	19
3.1.1.4 Vista de <i>Uses</i> do <i>Business</i>	21
3.1.1.5 Vista de <i>Uses Geral</i>	22
3.1.2 Vistas de Componentes e Conectores	24
3.1.2.1 Vista <i>Top Level SOA</i>	24
3.1.2.2 Vista de Componentes e Conectores do <i>Business</i>	26
3.1.2.3 Vista de Componentes e Conectores do <i>Web</i>	28
3.1.2.4 Vista de Componentes e Conectores do <i>GUI</i>	32
3.1.3 Vista de Alocação	35
3.2 Mapeamento entre Vistas	36
4 Bibliografia	38

Lista de Figuras

2.1	Caso de Uso 1 - Criar Torneio	7
2.2	Caso de Uso 2 - Registar Participante em Torneio	8
2.3	Caso de Uso 3 - Registar Resultado de Encontro	9
2.4	Caso de Uso 4 - Visualizar Calendário de Jogador	10
3.1	Vista de Decomposição do Business	15
3.2	Vista de Decomposição do Web	18
3.3	Vista de Decomposição do GUI	19
3.4	Vista de Uses do Business	21
3.5	Vista de Uses Geral	22
3.6	Vista Top Level SOA	24
3.7	Vista de Componente e Conectores do Business	26
3.8	Modelo de Dados	27
3.9	Vista de Componente e Conectores do Web	28
3.10	Vista de Interação Top Level SOA do RegistarResultado	30
3.11	Vista de Interação Top Level SOA do VerificarCalendario	31
3.12	Vista de Componente e Conectores do GUI	32
3.13	Vista de Interação Top Level SOA da Criação de Torneio	33
3.14	Vista de Interação Top Level SOA do RegistarParticipante	34
3.15	Vista de Alocação	35

Lista de Tabelas

3.1	Mapeamento entre a Top Level Soa e Vista de Alocação	36
3.2	Mapeamento entre a Multi-Tier do GUI e Vista de Alocação	36
3.3	Mapeamento entre a Multi-Tier do Web e Vista de Alocação	36
3.4	Mapeamento entre a Vista de Decomp. e C&C do Business	37
3.5	Mapeamento entre a Vista de Decomp. e Multi-tier do Web	37
3.6	Mapeamento entre a Vista de Decomp. e Multi-tier do GUI	37

Capítulo 1

Visão Geral e Mapeamento da Documentação

1.1 Propósito e *Scope* do SAD

Este documento descreve a arquitectura do sistema TornGest desenvolvido em CSS no ano lectivo 2017/2018. O TornGest é uma aplicação que permite a criação de torneios e a sua gestão. As funcionalidades do sistema inclui criar um torneio, adicionar um participante a um torneio, registar o resultado de um dado encontro e verificar o calendário de um jogador. Para isso foram feitas dois modos de apresentação diferentes, uma aplicação *desktop* com uma *GUI* que inclui as duas primeiras funcionalidades e um website que inclui as restantes duas.

1.2 Organização do SAD

Este SAD está dividido nos seguintes capítulos:

Capítulo 1: Neste capítulo dá-se uma introdução a este ficheiro, ao que está contido no mesmo e para quem é dirigido.

Capítulo 2: Neste capítulo fala-se das funcionalidades do sistema e dos requisitos.

Capítulo 3: Neste capítulo encontram-se as vistas com a sua documentação e a relação entre elas.

Capítulo 4: Neste capítulo encontra-se a bibliografia.

1.3 Representação dos *Stakeholders*

Para o TornGest existem os seguintes stakeholders:

A organização desportiva que pretenda comprar e usar a versão final do TornGest. A empresa que vai desenvolver o TornGest. Os utilizadores que vão utilizar a aplicação numa competição. A equipa de desenvolvimento, de manutenção e de qualidade.

A organização desportiva espera que o sistema seja reutilizável. A empresa que vai desenvolver espera que seja tudo feito em tempo e em *budget*. Os utilizadores finais esperam que a aplicação seja usável e que esteja disponível quando necessário. As várias equipas esperam que a arquitectura esteja bem definida e que o produto seja modificável e que vá de acordo com os princípios esperados.

1.4 Como uma vista é documentada

A secção 3 deste SAD contém todas as vistas arquitecturais dos tipos de módulos, componentes e conectores e alocação que nós realizámos durante este projeto. Cada vista é composta pelos seguintes pontos:

- Nome da vista – Contém o nome da vista.
- Descrição da vista – Breve descrição do uso e criação da vista.
- Catálogo de Elementos – Lista com descrições dos elementos que constituem a vista.
- Diagrama de Contexto – Esta secção mostra um diagrama de contexto que mostra o contexto da parte do sistema que é representado pela vista. Mostra também interações com entidades externas à vista.
- Mecanismos de variabilidade – Contém mecanismos de variabilidade arquitecturais descritos pela vista, incluindo uma descrição de quando e como esses mecanismos são utilizados.
- Racionalização – Contém uma explicação para decisões significativas de design que afetam a vista.

Capítulo 2

Background da Arquitetura

2.1 Descrição do Problema

O *TornGes* corresponde a um sistema que realiza a gestão de torneios de várias modalidades cujos jogadores e equipas de jogadores são classificados pelo sistema ELO.

Os jogadores, que pertençam ou não a equipas, têm encontros entre si dando origem a um resultado de vitória, derrota ou empate. Tendo em conta o resultado do encontro, são devidamente atribuídos os pontos a cada jogador e equipa, caso este pertença a uma.

O *TornGes* disponibiliza o seu serviço de gestão de torneios de modalidades-Elo a partir de diversos dispositivos com ligações à internet. O serviço disponibiliza duas interfaces que podem ser acedidas para interagir com o sistema. Ofereceu-se então ao utilizador a possibilidade de realizar esta interação com a aplicação recorrendo a um navegador *web* e a uma aplicação cliente *desktop*.

2.1.1 Contexto

2.1.1.1 Contexto Técnico

Os requisitos de qualidade correspondem a um dos principais fatores que definem o contexto técnico da arquitetura que foi escolhida. Encontram-se aqui posteriormente descritos os **Requisitos de Qualidade** utilizados para a escolha da arquitetura apropriada.

O ambiente técnico em que a arquitetura foi criada também desempenha um papel importante nas escolhas que foram tomadas. As técnicas de implementação utilizadas seguem um estilo *Object-Oriented* que à data da criação correspondiam às práticas *standard* da indústria para o desenvolvimento deste tipo de *software*.

2.1.1.2 Contexto do Ciclo de Vida do Projeto

O *TornGes* foi desenvolvido num processo de desenvolvimento de software iterativo. O *TornGes* corresponde a um dos componentes de um sistema final que apoia a organização de eventos competitivos em várias áreas para além realizar a gestão das competições, da logística, da divulgação e gestão financeira.

2.1.2 Requisitos Funcionais

Os gestores de torneios têm a possibilidade de interagir com o sistema através de cada um dos seguintes casos de uso.

2.1.2.1 Caso de Uso 1 - Criar Torneio

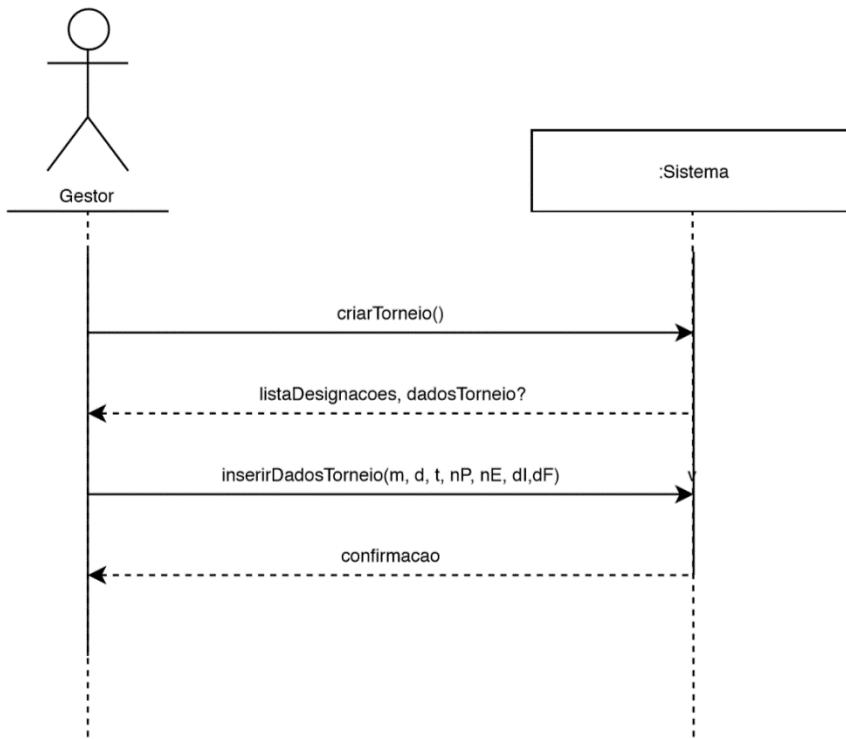


Figura 2.1: Caso de Uso 1 - Criar Torneio

Cenário: O gestor acede à aplicação e cria um torneio indicando a sua modalidade, designação, tipo, número de participantes, número de encontros entre cada par de oponentes, a sua data de início e duração.

Desencadear do evento: O gestor introduz as informações referidas anteriormente e pressiona “Criar Torneio”.

Breve descrição: A verificação do input é feita após ter sido pressionado “Criar Torneio”. Se o input tenha sido validado, então é apresentada uma mensagem de sucesso. Caso contrário, é devolvida uma mensagem indicando o erro da tentativa de criação do torneio.

Atores: Gestor.

Pré-condições: Nenhuma.

Pós-condições: Caso tenha ocorrido validação dos dados inseridos, é criado um torneio com a designação inserida.

Condições de exceção: A modalidade inserida é válida. Não existe outro torneio com a mesma designação, nem esta é null. O número de participantes e de encontros entre cada par de oponentes são maiores que zero e pares. Os torneios de equipas duram pelo menos dois dias e apanham um fim-de-semana.

2.1.2.2 Caso de Uso 2 - Registar Participante em Torneio

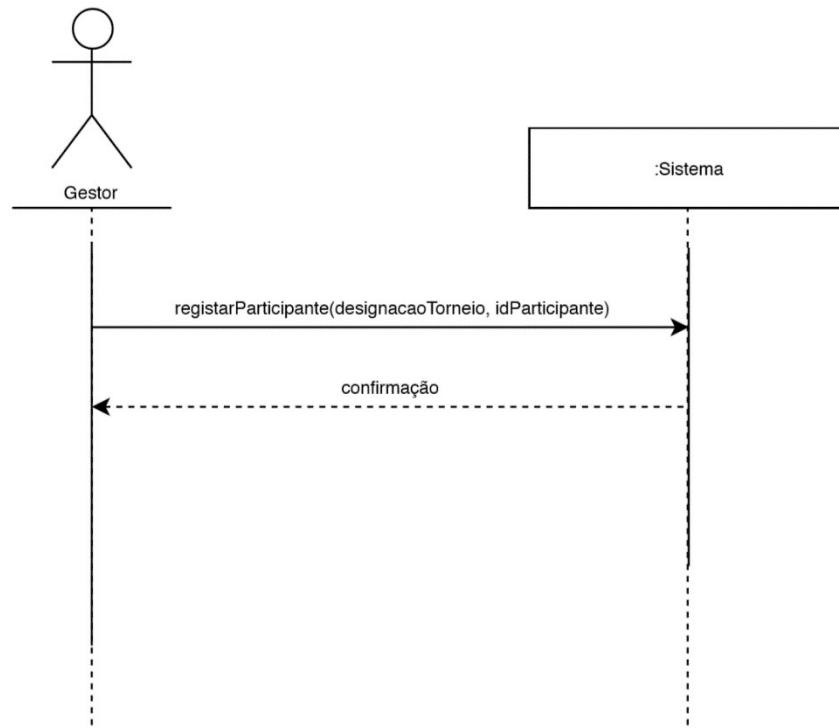


Figura 2.2: Caso de Uso 2 - Registar Participante em Torneio

Cenário: O gestor acede à aplicação e regista um participante, jogador ou equipa, num torneio, fornecendo a sua designação, o seu tipo e o número de inscrição do participante.

Desencadear do evento: O gestor indica o número de inscrição do participante, seleciona o torneio e pressiona “Concluir”.

Breve descrição: Quando o gestor conclui o registo é verificado pelo sistema se o número de inscrição inserido é válido, caso seja, então é devolvida uma mensagem de sucesso. Caso contrário, é devolvida uma mensagem de erro pelo que o gestor terá de introduzir outro número de utilizador.

Atores: Gestor.

Pré-condições: Nenhuma.

Pós-condições: Caso tenha ocorrido validação do número de inscrição do participante então o participante referido encontra-se inscrito no torneio.

Condições de exceção: Se o torneio onde se pretender registar o participante ainda se encontrar aberto. O número de inscrição fornecido é de um participante cujo tipo e modalidade corresponde aos do torneio. Caso o torneio seja do tipo equipa, a equipa participante tem de ter um número de elementos que está acima do limite imposto pela modalidade.

2.1.2.3 Caso de Uso 3 - Registar Resultado de Encontro

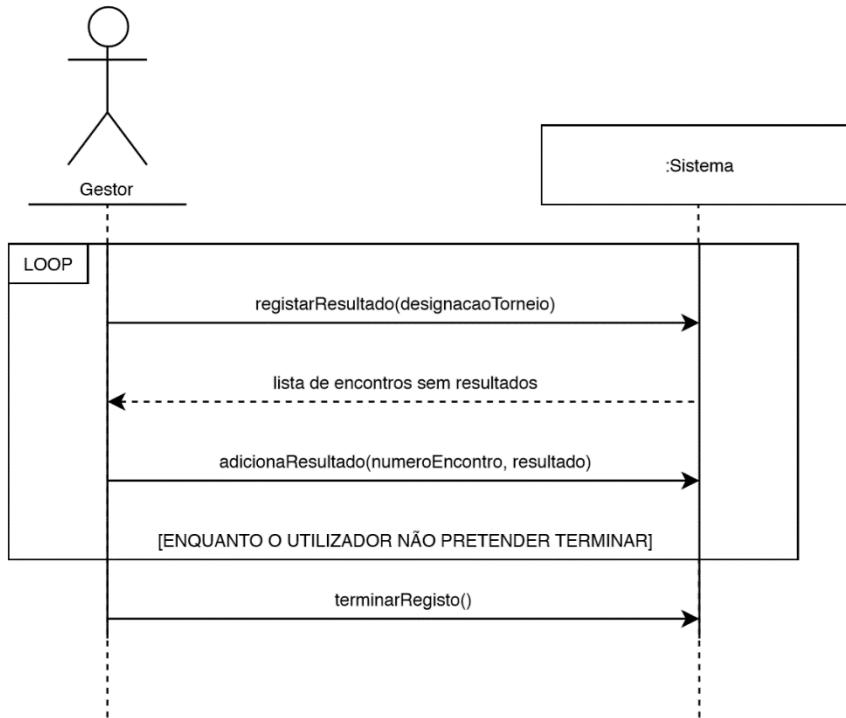


Figura 2.3: Caso de Uso 3 - Registar Resultado de Encontro

Cenário: O gestor acede à aplicação web e regista o resultado de um dos encontros de um torneio indicando a sua designação.

Desencadear do evento: O Registo do resultado é realizado após o gestor ter indicado o número de encontro e o seu resultado e clicado em “Submeter”.

Breve descrição: Após submeter o número de encontro e resultado, é verificado pelo sistema se este é válido. Caso seja válido então é aplicado o resultado e o utilizador pode realizar outro registo até pretender terminar.

Atores: Gestor.

Pré-condições: Nenhuma.

Pós-condições: Caso tenha ocorrido validação do número de encontro então este encontra-se com um resultado.

Condições de exceção: Se o número de inscrição for válido e se ainda não tem associado nenhum resultado.

2.1.2.4 Caso de Uso 4 - Visualizar Calendário de Jogador

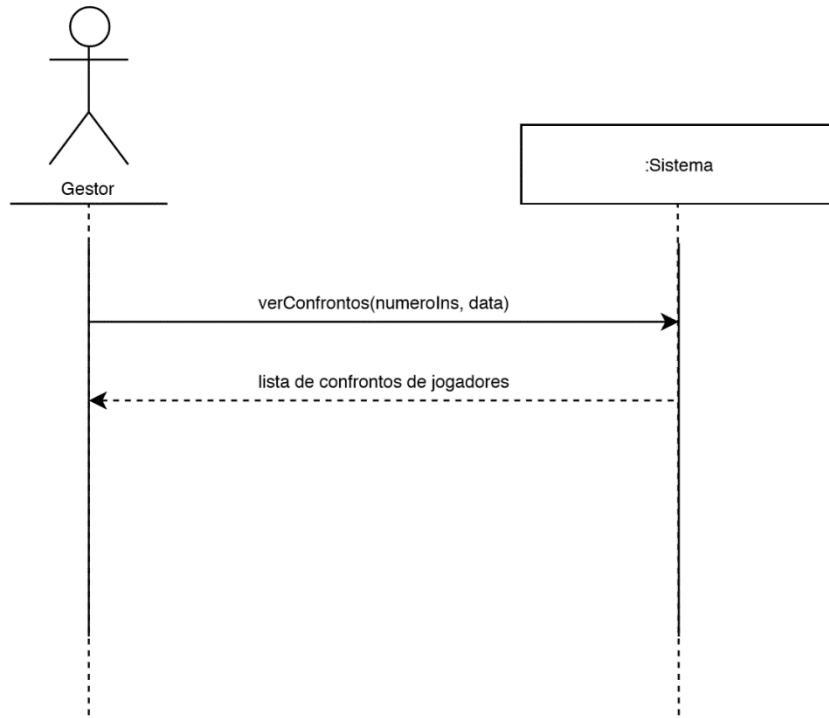


Figura 2.4: Caso de Uso 4 - Visualizar Calendário de Jogador

Cenário: O diretor acede à aplicação web e visualiza os confrontos de um jogador a partir de uma determinada data.

Desencadear do evento: O gestor indica o número de inscrição do participante e uma data e pressiona em “Visualizar”.

Breve descrição: Quando o diretor pressiona em Visualizar é verificado se o número de inscrição é valido e se a data se encontra num formato válido. Caso os dados sejam validados então é apresentada uma página web com todos os torneios onde o participante irá ter confrontos contra o seu adversário. Caso contrário, é dada uma mensagem de erro.

Atores: Diretor.

Pré-condições: Nenhuma.

Pós-condições: Nenhuma.

Condições de exceção: Se a data se encontrar num formato errado ou caso o número de inscrição seja inválido é lançada uma mensagem de erro ao diretor.

2.1.3 Requisitos de Qualidade

2.1.3.1 Desempenho

Caso de Uso 1:

Ambiente: A aplicação dispõe do seu conjunto de serviços que são disponibilizados através de chamadas remotas de uma aplicação com interface gráfica a vários utilizadores ao mesmo tempo.

Estímulo: Um gestor de torneios pretende realizar a criação de um torneio.

Resposta: A computação necessária para a criação do torneio demora menos de 10 segundos.

Caso de Uso 2:

Ambiente: A aplicação dispõe do seu conjunto de serviços que são disponibilizados através de chamadas remotas de uma aplicação com interface gráfica a vários utilizadores ao mesmo tempo.

Estímulo: Um gestor quer registar um participante.

Resposta: O resultado desse registo deve de ser notificado ao gestor em menos de 2 segundos sem contar com a latência da rede.

Caso de Uso 3:

Ambiente: A aplicação dispõe do seu conjunto de serviços que são disponibilizados na internet a vários utilizadores ao mesmo tempo.

Estímulo: Um gestor quer registar um resultado de um encontro entre um par de participantes.

Resposta: O resultado desse registo deve de ser notificado ao gestor em menos de 2 segundos sem contar com a latência da rede.

2.1.3.2 Fiabilidade:

Ambiente: Runtime.

Estímulo: Um gestor de torneios quer registar o resultado do torneio, mas como não obtém resposta até 10 segundos depois tenta realizar novamente o registo.

Resposta: Apenas o primeiro registo é contabilizado.

2.1.3.3 Modificabilidade

Caso de Uso 3:

Ambiente: Build time.

Estímulo: Vai ser adicionada um novo sistema de atribuição de pontos ao TornGest.

Resposta: Esta alteração não deve de ultrapassar as 2 pessoas-dia de esforço.

Ambiente: Build time.

Estímulo: Uma equipa de desenvolvimento vai alterar a quantidade de informação que é apresentada ao utilizador na visualização do calendário de um utilizador.

Resposta: Esta alteração não deve de ultrapassar as 2 pessoas-dia de esforço.

2.1.3.4 Usabilidade

Caso de Uso 1:

Ambiente: Aplicação desktop a correr.

Estímulo: Um gestor de torneios vai criar um novo torneio, mas engana-se e coloca uma designação de modalidade inválida.

Resposta: É apresentada uma janela com o erro de que a designação da modalidade inserida é inválida.

Caso de Uso 2:

Ambiente: Aplicação desktop a correr.

Estímulo: Um gestor de torneios registar um participante num torneio, mas esse torneio já está fechado.

Resposta: É apresentada uma janela com o erro de que o torneio está fechado.

Caso de Uso 3:

Ambiente: Aplicação web a correr.

Estímulo: Um gestor vai registar o resultado do encontro que aconteceu, mas esquece-se de colocar a designação.

Resposta: A invalidade da designação é apresentada na página, na secção mensagens.

Caso de Uso 4:

Ambiente: Aplicação web, página para visualizar o calendário de um jogador.

Estímulo: Um utilizador vai consultar o calendário numa data errada.

Resposta: A invalidade da data é apresentada na página.

2.1.3.5 Testabilidade

Ambiente: A aplicação dispõe os seus elementos da camada de serviços convenientemente isolados utilizando *Separation of Concerns*.

Estímulo: Um *tester* pretende testar cada um dos requisitos funcionais.

Resposta: O grau esforço necessário para realizar os testes a cada um dos casos de uso deverá de ser 1 pessoa dia.

2.1.3.6 Interoperabilidade

Ambiente: Design time.

Estímulo: A equipa de desenvolvimento pretende desenvolver uma aplicação móvel para o TornGest que realize todos os requisitos funcionais.

Resposta: O esforço necessário para realizar a implementação das chamadas aos serviços remotos deverá de ser inferior a 2 pessoas dia por requisito funcional.

2.1.3.7 Potencial de Reutilização:

Ambiente: A aplicação dispõe os seus elementos da camada de serviços convenientemente isolados utilizando *Separation of Concerns*.

Estímulo: A equipa de desenvolvimento pretende integrar o tornest com um módulo de gestão de desportos.

Resposta: O grau esforço necessário para realizar a integração de cada um dos módulos existentes no sistema não deverá de exceder os 4 pessoa dia, 1 pessoa para cada unidade de software.

2.2 *Background* da Solução

2.2.1 Solução Arquitetural

Para a arquitetura deste sistema foram tomadas várias decisões de *design*. Primeiramente foi utilizado uma arquitetura *N-tier Client Server* na medida que é benéfico encapsular conjuntos de serviços em *Tiers* e promove escalabilidade e disponibilidade. É possível observar esta decisão de desenho na organização dos vários tiers em que cada um disponibiliza um conjunto de serviços, temos o *Tier* do cliente que pode conter o *browser web* no caso da aplicação *web* ou a aplicação *desktop* no caso do *GUI*, o tier de apresentação com todos os componentes que tratam de ambas as interfaces e que serão vistos em detalhe mais a frente nas vistas de componentes e conectores, o *Tier* de *EJB's* que funcionam como intermediários entre a apresentação e o *backend*, o *Tier* de lógica de negócio e por fim o tier com a base de dados. Os vários clientes que lancem o sistema ligam-se a um servidor *Wildfly* numa arquitetura cliente-servidor cuja ligação é algo que é abstraído pelo *Java EE*.

Foi também escolhida a decisão de implementar uma arquitetura com repositório de dados partilhado uma vez que é necessário guardar todos os dados sobre jogadores, torneios, encontros, equipas, entre outros que deve de estar partilhado por vários clientes que precisem de manipular os dados como registar resultados de encontros ou criar jogadores. Também foi utilizada uma estrutura *SOA* (*Service oriented architecture*) com o uso da camada de serviços e das interfaces remotas (*Service Broker*), *beans* de sessão sem estado, que aceitando tipos de dados primitivos permite que os consumidores (*Service Consumers*) sejam capazes de utilizar os serviços (*Service Providers*) sem conhecer detalhes de implementação no *backend*.

Em relação a padrões de desenho foram utilizados o *Data Mapper* porque havia necessidade mapear objetos do mundo do Java para os objetos no mundo relacional de modo a inserir na base de dados *MySQL*. É um padrão adequado visto que o modelo de domínio é algo complexo e aumenta o *decoupling* entre as classes do sistema e o mundo relacional. Foi também utilizado o paradigma *Model View Controller* para ambas as interfaces de utilizador do sistema: a aplicação *web* e o *GUI*. Para a aplicação *web* existe uma classe *FrontController* que trata de receber os pedidos encaminhando-os para as páginas JSP respetivas aplicando modelos para obter os dados da introdução do utilizador. No cliente *GUI* existe um controller por vista, sendo esta guardada num ficheiro *FXML*.

Capítulo 3

Vistas Arquiteturais

3.1 Vistas

3.1.1 Vistas de Módulos

3.1.1.1 Vista de Decomposição do *Business*

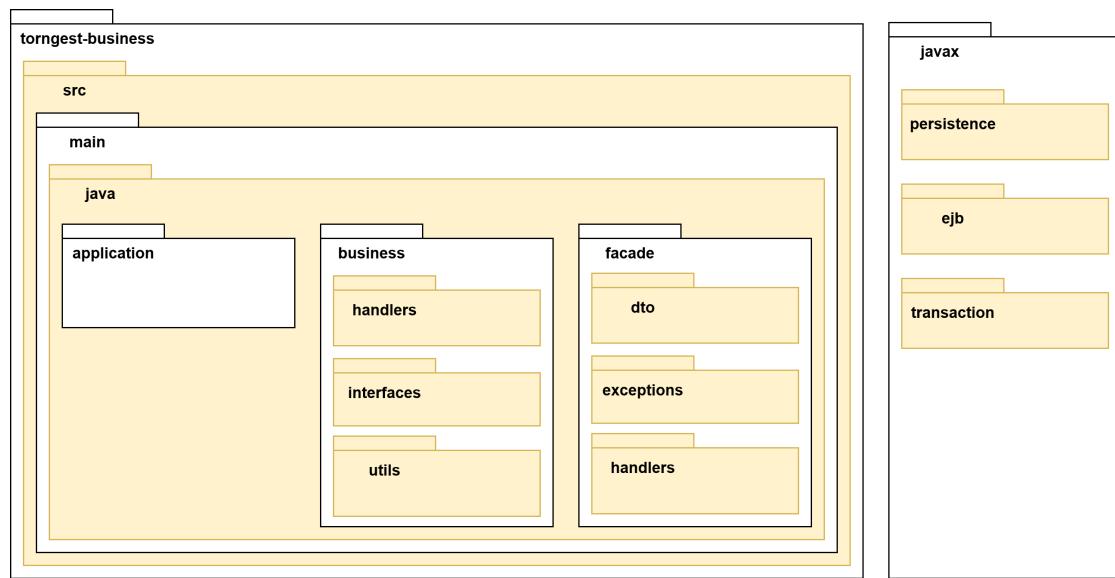


Figura 3.1: Vista de Decomposição do Business

Descrição da Vista:

O objetivo desta vista de módulos é o de representar todas as unidades de implementação que foram usadas num dos principais módulos do sistema. O módulo de torngest-business corresponde ao módulo que é responsável pela unidade de processamento da lógica computacional do sistema. A divisão em vários diferentes módulos do torngest-business permite que seja possível alcançar uma maior modularização do sistema de modo a que seja possível adicionar e remover requisitos funcionais sem influências no resto do sistema. Os requisitos funcionais que se encontram implementados na lógica do negócio são os seguintes:

- Realizar as ações propostas vindas de um cliente *desktop* para a Criação de um Torneio e Registo de um Participante num torneio existente na base de dados, através de um *Torneio Web Service*.
- Providenciar uma forma do utilizador através de um Cliente Web de Visualizar Calendário de um Jogador através de um *Calendário Web Service*.
- Realizar também o Registo de um Resultado recorrendo também ao *Torneio Web Service*.

- Finalmente, após realização das tarefas invocadas, realizar o output da tentativa de realização das mesmas.

Visualização geral da Vista:

A vista com o conjunto de módulos apresentado é suficientemente completa para realizar a representação necessária dos Casos de Uso propostos. Cada módulo é responsável por uma parte do sistema e não existe trabalho duplicado entre os módulos. A vista de módulos apresentada corresponde à parte do *backend* de uma arquitetura *N-Tier Client-Server* onde os clientes, por exemplo *Web*, acedem aos serviços como através de uma camada de fachada correspondente aos serviços que são prestados pelo sistema e que se encontram na vista da *application*.

Influência da arquitetura na Vista:

Para além de uma arquitetura *N-Tier Client-Server* foi também aplicada uma arquitetura *Database* onde o servidor é responsável por realizar as alterações na base de dados local. O modo como o *business* foi implementado segue uma forma em 3 camadas. Onde temos uma camada de aplicação responsável pela prestação de serviços, temos um modelo de domínio que executa as operações do sistema e temos finalmente uma base de dados centralizada, que corresponde à terceira camada de acesso aos dados, onde são realizadas escritas e leituras. Como foi recorrido a *EJB's* foi possível abstrair grande parte da implementação das tarefas de escritas e leituras na base de dados.

Diagrama de Contexto: N/A.

Mecanismos de Variabilidade: N/A.

Catálogo de Elementos:

application: O módulo application representa o ponto de entrada para a lógica do nosso sistema. Neste módulo encontram-se os serviços que podem ser acedidos externamente através de Web Services para realizar a interação com o sistema. Este módulo permite criar uma barreira dos serviços que podem ser acedidos.

business: Encontra-se implementado neste módulo toda a lógica computacional do sistema. Este módulo encontra-se também composto por três sub-módulos. Os **handlers** que contêm as operações necessárias para realizar os casos do uso do sistema. As **interfaces** que apresentam respetivamente as interfaces implementadas pelos handlers para definição das operações dos casos de uso. Os **utils** onde se encontra toda a lógica utilitária a ser utilizada no sistema.

facade: O módulo facade é essencial para realizar o ponto de encontro entre a lógica do sistema e o que será entregue ao serviço que pretende realizar a operação. Este módulo é realiza uma abstração de como o sistema se encontra composto. O **facade** pode ser decomposto em três sub-módulos.

As **exceptions** onde se encontra os tipos de erros que poderão ser lançados durante uma tentativa de execução de uma operação no sistema.

Os **handlers** que contêm a definição das operações que correspondem aos serviços que são prestados pelo sistema.

Os **dto's** que contem as classes com a apenas a informação necessária a ser entregue pelo sistema ao cliente que tentou executar uma operação no serviço. Os *Data Transfer Object* utilizados são os seguintes.

EncontroDTO: Este corresponde a um tipo que é devolvido na operação `getEncontrosTorneio` e que indica todos os encontros entre dois jogadores.

```
int numeroEncontro
String nomeJogador1
String nomeJogador2
```

JogadorShortDTO: Este tipo é utilizado na operação getJogadores e permite listar todos os jogadores existentes no sistema torngest.

```
int id
String nome
```

TorneioShortDTO: Este tipo é utilizado na operação getAllTorneios e permite listar todos os torneios existentes no sistema torngest.

```
String designacaoTorneio
```

VisualizarEncontroDTO: Este tipo é utilizado na operação verConfrontos e permite listar todos os torneios em que dois jogadores joguem a partir de uma determinada data.

```
String designacaoTorneio
String dataInicio
JogadorDTO jogadorCorrente
JogadorDTO jogadorAdversario
int qtdEncontros
```

3.1.1.2 Vista de Decomposição do Web

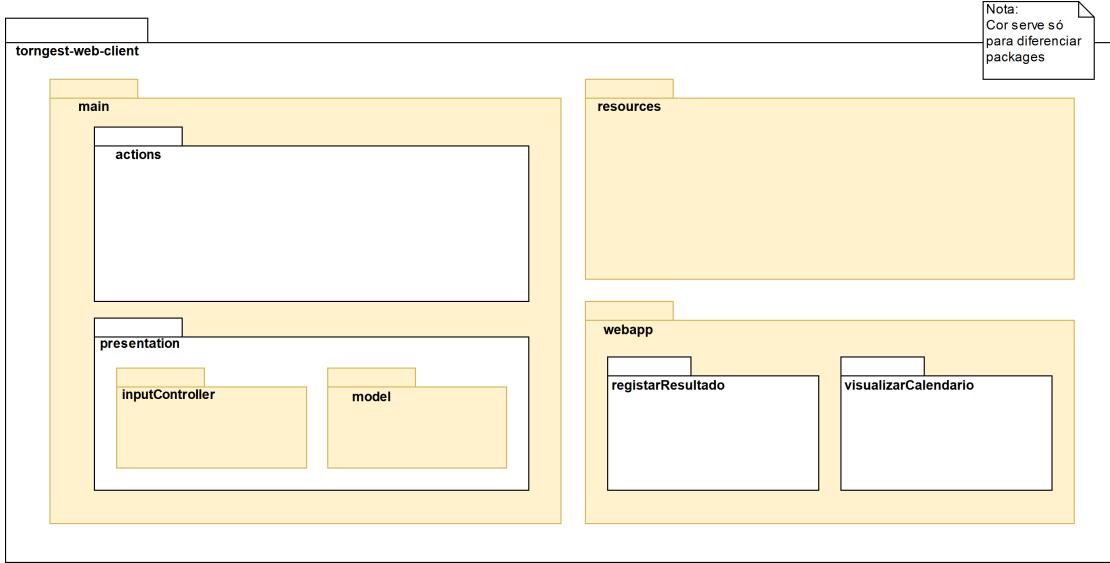


Figura 3.2: Vista de Decomposição do Web

Descrição da Vista:

Esta vista descreve a decomposição em módulos da *web application*, onde é tratada toda a parte relacionada com a aplicação *web*.

Catálogo de Elementos:

torngest-web-client: Este módulo inclui toda a lógica de apresentação da aplicação *web* do *torngest*, onde duas das quatro funcionalidades residem, registar resultado e visualizar calendário de um jogador.

src: Todo o código encontra-se dentro deste módulo, os ficheiros java para o processamento de *input*, os ficheiros html e jsp para apresentação das páginas e ficheiros de configuração.

actions: Nas actions encontra-se todo o código java que trata do input recebido pelo utilizador. Isto é conseguido através do uso de *HttpServlets*, obtendo os valores introduzidos pelo utilizador através do *model* respectivo. Chama os métodos dos serviços da camada de negócio através de *Stateless Session Beans*.

presentation: O módulo presentation é usado pelo actions, para obter os valores dos modelos, e é composto por dois módulo, apresentados a seguir.

inputController: Aqui encontra-se o *Front Controller*, que recebe todas as chamadas *HTTP* e reenvia para a action correspondente.

model: Neste módulo estão guardados todos os dados que o utilizador vê na página *web* e que ele próprio insere na página.

resources: Nos resources está o ficheiro de configuração da *web app*, que diz que actions correspondem a que páginas.

webapp: Neste módulo estão dois submódulos que contêm os ficheiros jsp para apresentar a página, e para guardar os valores inseridos no *model*.

registarResultado: O ficheiro jsp da operação de registar um resultado encontra-se aqui, em conjunto com um ficheiro jsp que mostra os torneios correntes.

visualizarCalendario: O ficheiro jsp da operação de visualizar o calendário de um jogador encontra-se aqui, em conjunto com um ficheiro jsp que mostra os confrontos

desse jogador.

Diagrama de Contexto: N/A.

Mecanismos de Variabilidade: N/A.

3.1.1.3 Vista de Decomposição do *GUI*

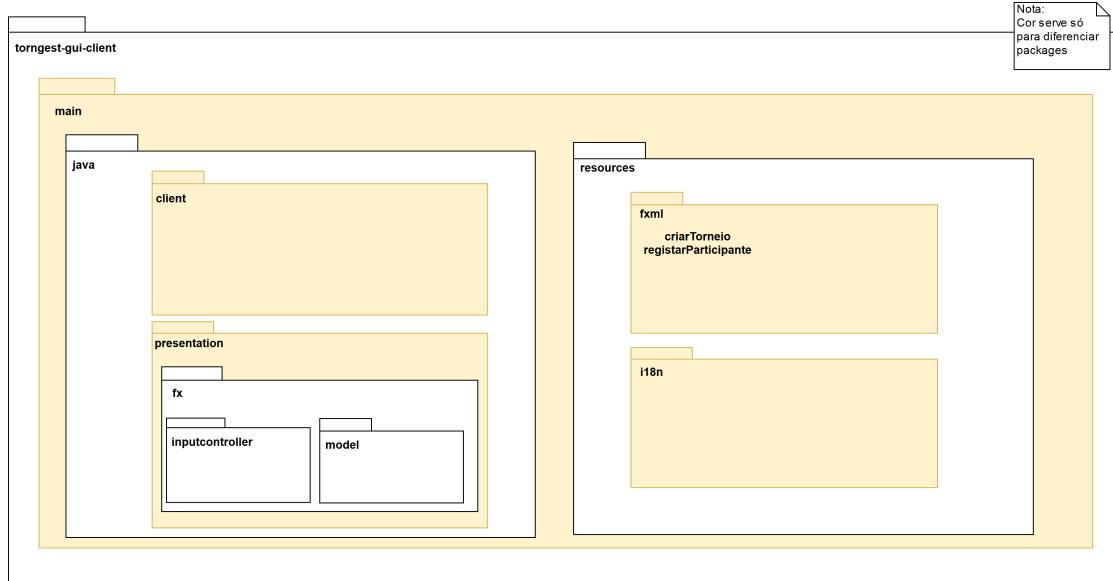


Figura 3.3: Vista de Decomposição do GUI

Descrição da Vista:

Esta vista corresponde à decomposição em módulos da parte da *GUI* do *TornGes* e mostra como está dividido em pacotes, ou seja, em unidades de código. Foram omitidos os pacotes mais abrangentes como o `src`. Esta vista tem como objetivo ajudar a entender a organização da implementação desta parte do sistema.

Catálogo de Elementos:

tornkest-gui-client: Este módulo inclui toda a lógica de apresentação da aplicação *GUI* do tornkest onde existem as funcionalidades de criar um torneio e de registrar um participante e é o módulo em que esta vista incide.

java: Esta aplicação *desktop* corre na *framework* de desenvolvimento *Java EE* e foram colocadas aqui todas as unidades de código.

client: Este módulo contém o código necessário para correr o cliente da *GUI*, nomeadamente chama o Startup no pacote presentation que faz a inicialização dos controllers e das views.

presentation: Este módulo contém toda a parte que diz respeito ao arranque da interface bem como à articulação dos controladores e o código que diz respeito ao modelo de dados.

resources: Aqui são colocados todo o código relativo à apresentação do layout da *GUI* bem como outros recursos necessários ao sistema.

FXML: Aqui foram colocados os ficheiros fxml que correspondem às páginas que serão apresentadas na janela da interface.

i18n: Contém os ficheiros com recursos necessários ao sistema, strings de erros ou de informações a apresentar.

Diagrama de Contexto: N/A.

Mecanismos de Variabilidade: N/A.

Background da Arquitetura:

Para o caso do componente da *GUI* do *TornGes* foi utilizado como padrão principal de organização o padrão *MVC* ou *Model View Controller* e isso reflete-se na organização das unidades de código. Verificamos que dentro do pacote *presentation* localizado no pacote *java* está contido um outro pacote o *fx* que por sua vez tem lá a componente que corresponde ao *controller* no pacote *inputcontroller* e dentro da pasta *model* o componente correspondente ao modelo.

Já no pacote *fxml* temos a parte correspondente à *view* com ficheiros *fxml* onde cada um representa a página *web* referente a cada funcionalidade neste caso a criação do torneio e o registo do participante. A utilização do padrão *MVC* tem impacto no projeto na medida que há uma *Separation of Concerns* da interface do resto da lógica e dados do sistema.

3.1.1.4 Vista de *Uses* do *Business*

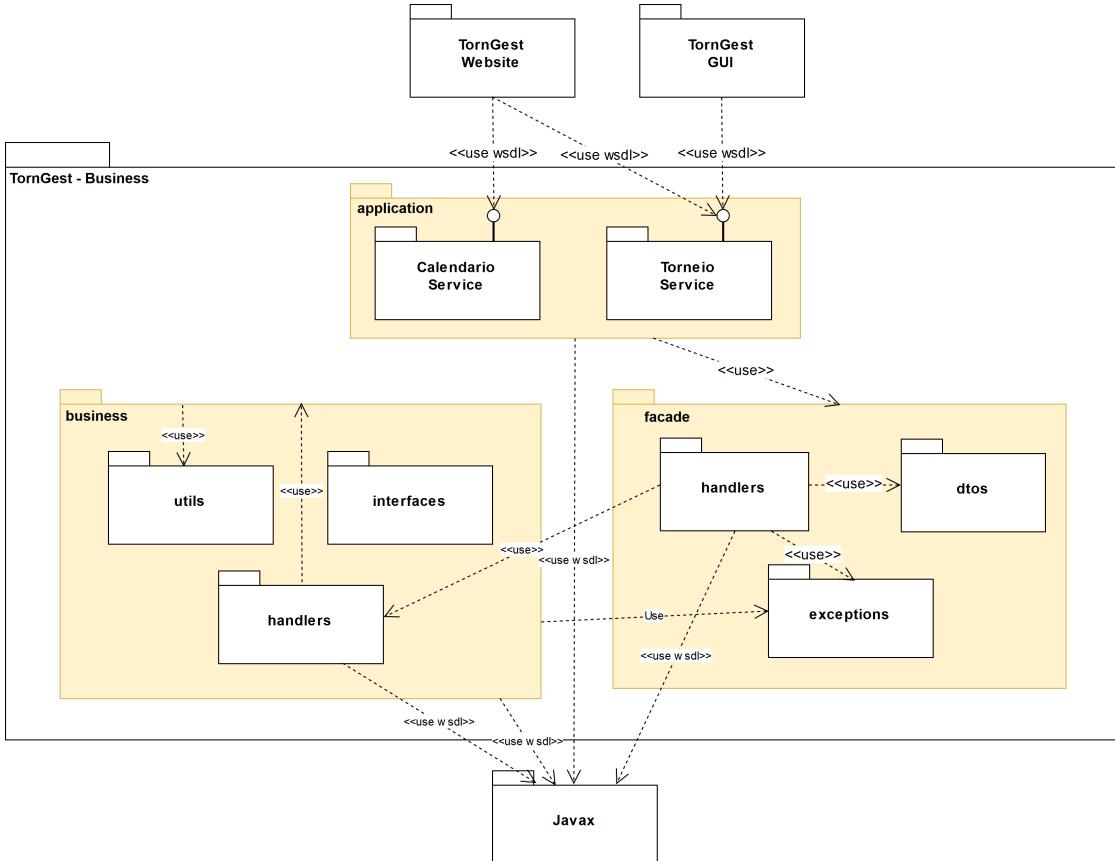


Figura 3.4: Vista de Uses do Business

Descrição da Vista:

O objetivo desta vista de *uses* é o de representar todas as relações de *uses* no *business*, como e onde é que os clientes externos comunicam com o *business*, e com que serviços externos o *business* comunica.

Catálogo de Elementos:

application: O módulo application representa o ponto de entrada para a lógica do nosso sistema. Neste módulo encontram-se os serviços que podem ser acedidos externamente através de Web Services para realizar a interação com o sistema. Este módulo permite criar uma barreira dos serviços que podem ser acedidos. O business oferece dois serviços: CalendarioService e Torneio servisse. Este módulo usa todos os módulos do facade: implementam os handlers, usam dtos para enviar os dados e usam as exceções para envio de mensagens de erro. Usa também um módulo externo, o javax.

business: Encontra-se implementado neste módulo toda a lógica computacional do sistema, contendo as classes que estão a ser persistidas na base de dados. Este módulo encontra-se também composto por três sub-módulos. Os handlers que contêm as operações necessárias para realizar os casos do uso do sistema. As interfaces que apresentam respetivamente as interfaces implementadas pelos handlers para definição das operações dos casos de uso. Os utils onde se encontra toda a lógica utilitária a ser utilizada no sistema. Os handlers usam o módulo externo javax.

facade: O módulo facade é essencial para realizar o ponto de encontro entre a lógica do sistema e o que será entregue ao serviço que pretende realizar a operação. Este módulo é necessário para realizar uma abstração de como o sistema se encontra composto.

O **facade** pode ser decomposto em três sub-módulos. As exceptions onde se encontra os tipos de erros que poderão ser lançados durante uma tentativa de execução de uma operação no sistema.

Os **handlers** que contêm a definição das operações que correspondem aos serviços que são prestados pelo sistema. Estes usam vários módulos: os dto's, as exceções, o módulo handler do business e o módulo externo javax.

Os **dto's** que contêm as classes com a apenas a informação necessária a ser entregue pelo sistema ao cliente que tentou executar uma operação no serviço.

facade: Este módulo externo é usado por vários dos módulos contidos no business do TornGest e é usado para dizer que esta vai usar *EJB's* e vai ser gerida por um *container*. É também usada para dizer que os Serviços oferecidos vão ser *Web Services* e para que todo o *container* trate das interacções com a base de dados.

3.1.1.5 Vista de *Uses Geral*

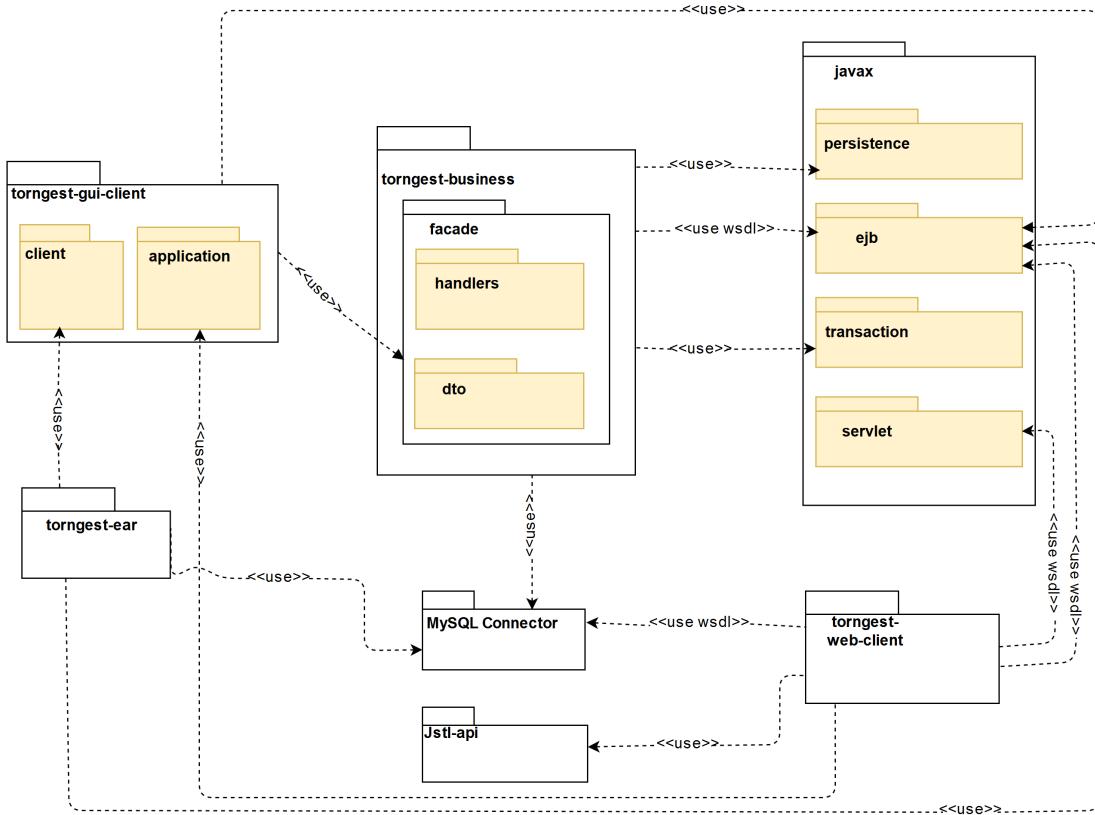


Figura 3.5: Vista de *Uses Geral*

Descrição da Vista:

Esta vista corresponde a uma vista geral dos módulos do projeto segundo a relação usa (“uses”) entre os mesmos. Tem o objetivo de dar uma panorâmica geral dos módulos do projeto e as dependências entre eles de modo a sistematizar o sistema.

Catálogo de Elementos:

torngest-gui-client: Este pacote contém todas as unidades de código e módulos pertencentes à funcionalidade do cliente de interface gráfica do sistema. Nesta interface é possível criar um torneio e registar um participante.

torngest-business: Este pacote contém todas as unidades de código e módulos pertencentes à lógica de negócio do sistema TornGest. Como tal, este módulo será utilizado por clientes que queiram utilizar esta lógica de negócio como o *GUI* e a *WebApp*.

torngest-ear: Este pacote liga todas as funcionalidades do sistema gerando um ficheiro que depois pode ser instalado em outras máquinas. O ficheiro *ear* contém o *war* da aplicação *web* em *torngest-web-client*, *jar's* para *Enterprise Java Beans* e outros componentes como um *resource adapter*.

torngest-web-client: Contém todo o código pertencente à funcionalidade da aplicação *web* deste sistema. Este módulo utiliza *servlets* e *JST*'s gerando um *war*. É possível registar um resultado de um torneio bem como visualizar o calendário de um jogador.

MySQL Connector: Contem o controlador necessário para ligar o sistema a uma base de dados MySQL.

jstl-api: Contém as API's referentes as páginas *JSP* que são utilizadas pela aplicação *web* do *TornGest*.

javax.persistence Este módulo é externo à aplicação e contém as API's relacionadas com a persistência dos dados que é utilizada no sistema. **javax.ejb** Este módulo é externo à aplicação e contém as API's relacionadas com os *Enterprise Java Beans* que são utilizados no sistema. **javax.transaction** Este módulo trata das transações das operações feitas no *TornGest*. **javax.servlet** Este módulo contém API's dos servlets que são utilizados pela aplicação *web*.

Diagrama de Contexto: N/A.

Racionalização:

A framework *Java EE* foi escolhida para este sistema por gerir facilmente as entidades com os *Enterprise Java Beans* bem como a adaptabilidade à persistência com o JPA (*Java Persistence API*). Facilita também a *Separation of Concerns* de cada componente do sistema.

3.1.2 Vistas de Componentes e Conectores

3.1.2.1 Vista Top Level SOA

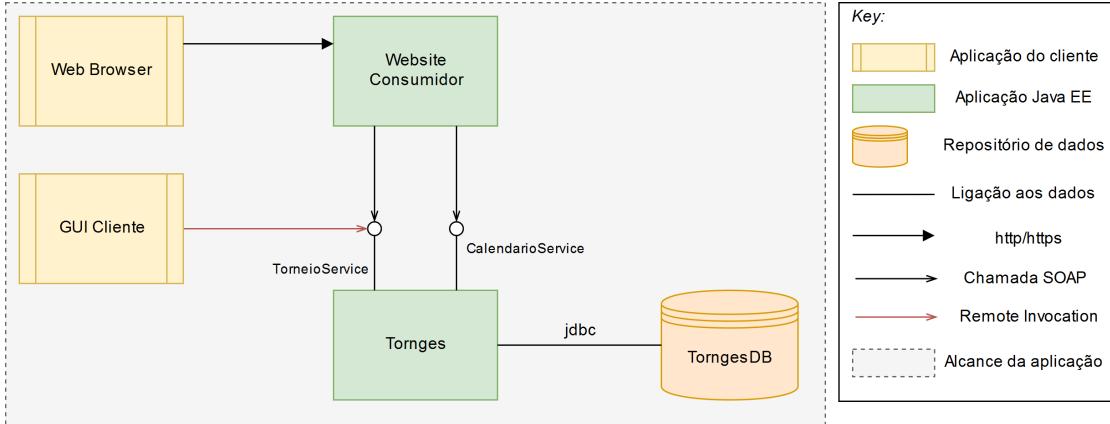


Figura 3.6: Vista Top Level SOA

Descrição da Vista:

Esta vista apresenta uma visão geral do sistema com as interações que ocorrem entre a aplicação e os utilizadores dos *Web Services*.

Element Catalog:

Web Browser

Este componente representa a interface gráfica da aplicação que se encontra a correr no *Web Browser* do Cliente. A aplicação *web*, permite a um Gestor do *Tornges* de realizar várias operações sobre o sistema, como por exemplo, visualizar o calendário de um determinado jogador e também registar o resultado de uma partida entre um par de jogadores.

Website Consumidor

O *Website Consumidor* segue um nível aplicacional multicamadas usando o *Java EE*. A visualização é feita recorrendo a um index.html, a JSPs e a view models. A sua responsabilidade primário é o de processar os pedidos *HTTP* que vêm através das ações realizadas pelos utilizadores enquanto navegam e interagem com a aplicação *web*. Os pedidos são realizados através de *Remote Invocation*.

GUI Cliente

O cliente *GUI* foi desenvolvido recorrendo a uma variante do *MVC* recorrendo a propriedades e *data binding*. Os controladores foram criados usando o *Page Controller* e o acesso à camada de negócios foi feita utilizando injeção de *session beans* remotos, seguindo no entanto as regras do *Java EE* de *application client container*.

Tornges

O *Tornges* é uma aplicação de gestão de torneios. Corresponde a uma aplicação desenvolvida em *Java EE* e que disponibiliza a componentes externos a interação com o sistema através de *web services*. Internamente é composto por *EJB's* que comunicam entre si através de *Java calls*. A vista de *Tornges C&C Business View* apresenta uma descrição mais detalhada dos componentes internos do *Tornges*.

O *Tornges* disponibiliza para as aplicações externas as seguintes interfaces:

- **TorneioService** – usado para realizar as operações enquadradas no Caso de Uso 1, registar um torneio, e Caso de Uso 2, registar um participante num torneio.
- **CalendarioService** – usado essencialmente para duas operações: visualizar os calendarios dos jogadores e os seus encontros e registar os resultados dos encontros de uma determinada partida entre um par de jogadores.

TorngesDB

A base de dados relacional é utilizada para armazenar todas as informações que existem no Modelo de Domínio, tal como os Jogadores, as Equipas, os Torneios, os seus Encontros e Confrontos com o seu respetivo Desfecho de Encontro. O servidor de base de dados utilizado foi o MySQL.

Mecanismos de Variabilidade

A utilização do padrão *Front Controller* permite uma maior facilidade e rapidez na implementação de novas páginas, sendo que não é necessário criar um *controller* por cada página dos serviços web. O ficheiro *app.properties* permite definir as páginas existentes e que são executadas recorrendo a *Actions*. Foi utilizado o padrão *DTO* para abstrair os clientes dos dados que estão a receber.

3.1.2.2 Vista de Componentes e Conectores do *Business*

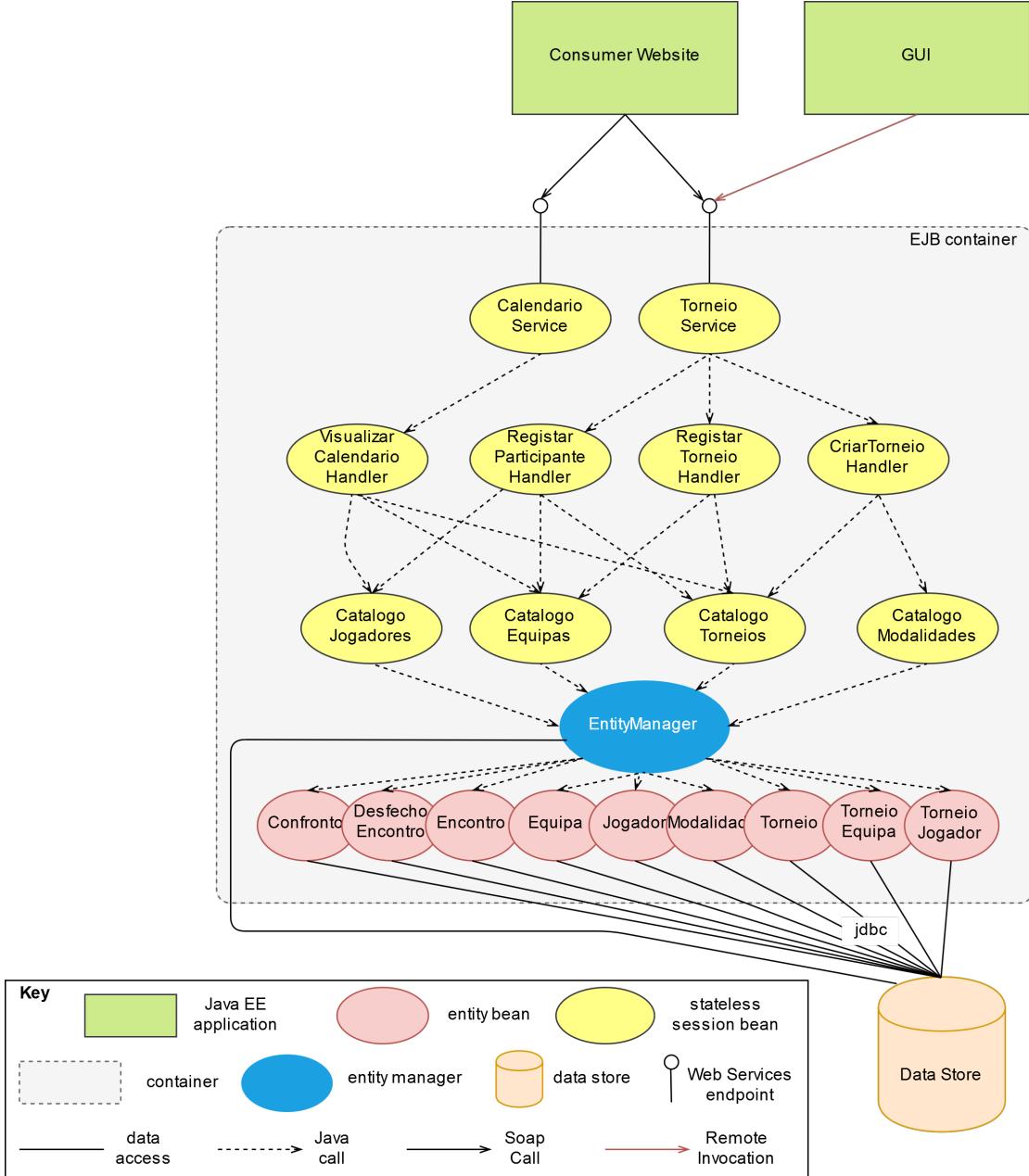


Figura 3.7: Vista de Componentes e Conectores do Business

Descrição da Vista

Esta vista descreve como se encontram conectados cada um dos elementos do sistema TornGest. As ligações entre os *entity beans* foram ocultadas para melhorar a visualização geral da vista. Recorrendo a anotações do Java EE é possível concretizar esta implementação.

Podemos verificar na vista que os serviços podem ser acedidos através de dois *Stateless Session Bean*, o CalendarioService cujos serviços são prestados apenas a aplicações *web* enquanto que os serviços do TorneioService são prestados tanto à aplicação *web* quanto ao *GUI*. Estes serviços recorrem então a Handlers que que através de chamadas Java acedem também aos Catalogos. As dependências fracas entre os Handlers e

os *Entity Beans* foram ocultadas. Cada Catálogo recorre a um *Entity Manager* para realizar operações sobre a base de dados dos objetos que lá se encontram armazenados. O *Entity Manager* é obtido pelo Catálogo através de de injeção da dependência feita pelo *container*.

Catálogo dos Elementos

Stateless Session Beans: Estes beans foram utilizados uma vez que não mantêm sessão com o cliente. Quando um cliente se conecta é criada uma instância com um determinado estado que apenas se prolonga durante a invocação. Como não é necessário manter sessão, é possível escalar o nosso serviço e ter múltiplos clientes sem ter problemas de desempenho do nosso servidor.

Entity Bean: As classes anotadas como *Entity Beans* comprovam que elas representam dados que necessitam de ser armazenados numa base de dados. Através do *Entity Manager* estes beans podem ser manipulados e armazenados na base de dados. Os beans apresentados na vista representam a parte de negócios do nosso sistema.

Entity Manager: O *Entity Manager* realiza a ligação entre os Beans de sessão e a base de dados. Este é injetado pelo *container* em cada um dos catálogos e permite realizar a manipulação dos *Entity Bean* e armazená-los.

Data Store: Representa a base de dados relacional que armazena cada um dos objetos convertidos pelo *Data Mapper*. O diagrama seguinte demonstra o Modelo da Persistência de dados na base de dados.

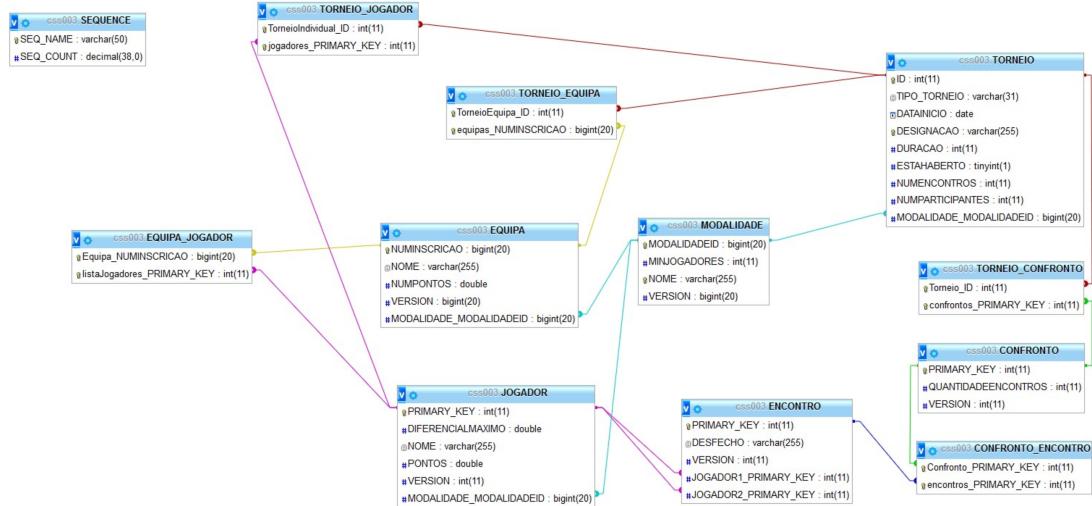


Figura 3.8: Modelo de Dados

3.1.2.3 Vista de Componentes e Conectores do Web

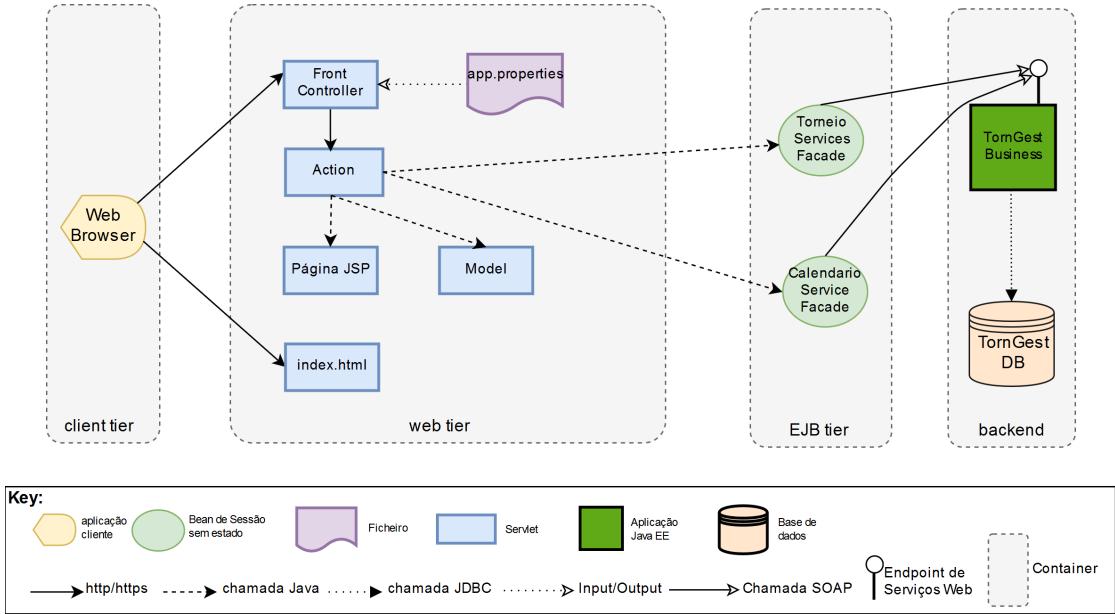


Figura 3.9: Vista de Componentes e Conectores do Web

Descrição da Vista

Esta vista descreve as principais unidades de computação e repositório e a maneira como se relacionam em tempo de execução da aplicação *web* do *TornGest*. Foca-se também nos protocolos e do meio de comunicação entre cada unidade de computação.

Catálogo dos Elementos

Web Browser: Este componente representa a interface gráfica da aplicação a correr no *Web Browser*. É uma interface bastante simples pelo que só usa HTML com formulários para a inserção de dados. Através do *Web Browser* um utilizador do *TornGest* abre o *website* e consegue registar o resultado de um torneio e também visualizar o calendário de jogos do jogador.

index.html: Esta é a página *web* de início quando um utilizador do *TornGest* arranca com a aplicação *web* e que também contém as ligações para as restantes páginas da aplicação.

Front Controller: Este *Servlet* faz parte do *Java EE*, e corresponde ao componente do *Controller* do modelo *MVC* que está encarregue de receber os pedidos que vêm do utilizador. Este *Servlet* utiliza o ficheiro *app.properties* para selecionar a ação que o utilizador quer desempenhar.

Action: Este componente representa cada ação que é possível realizar no website e faz o carregamento da página correspondente é mostrada ao utilizador no seu *browser web*. De salientar que na vista as ações foi abstraídas por este componente “Action” mas no *TornGest* existem as seguintes ações:

- **RegistrarResultado:** Esta ação corresponde à funcionalidade de registrar um resultado de um torneio. Em caso de sucesso retorna o JSP “registerResultado.jsp” com opção de colocar o número do encontro bem como o resultado do mesmo e em caso de insucesso retorna a mesma página, mas com uma mensagem de erro.

- **ProcessarRegistoResultado:** Esta ação é responsável pelo registo efetivo do resultado que foi inserido na página de registar o resultado e de retornar a página “visualizarTorneios” com o desfecho dos jogos já com o resultado inserido tido em mente. Caso não haja torneios, encontros ou desfechos de encontros é enviada uma mensagem de erro e fica na mesma página.
- **UnknownAction:** Esta ação corresponde a um caso de uso onde o utilizador insira qualquer input na barra de URL que não seja as ações suportadas carregando a página JSP “unknownAction.jsp” que simplesmente apresenta uma mensagem de ação desconhecida.
- **VerConfrontos:** Esta ação ocorre quando o utilizador inseriu os atributos do número de inscrição e da data na página de visualizar o calendário. Quando a data é correta é apresentado o JSP “verConfrontos.jsp” e quando é incorreta é apresentada a mesma página com uma mensagem de erro que vem do business.
- **VisualizarCalendario:** Esta ação corresponde à funcionalidade de visualizar o calendário de jogos de um jogador. Esta ação carrega o JSP “visualizarCalendario.jsp” onde o utilizador pode inserir o número de inscrição e a data. Ao inserir estes dois campos é despoletada a ação VerConfrontos e respetivo JSP.
- **VisualizarTorneios:** Esta ação é despoletada quando o utilizador quer executar a operação de visualizar o calendário e insere o número de inscrição e a data. São apresentados os torneios para o dado calendário do jogador.

Cada ação escolhe uma página de acordo com a ação entendida e preenche o mesmo com os dados do *Model* correspondente.

Página JSP: Para simplificar a vista foram abstraídas todas as páginas JSP num componente “Página JSP”. No TornGest para a funcionalidade de registar um resultado existem os JSP’s “registarResultado”, “visualizarTorneios” e para a funcionalidade de visualizar o calendário existem os JSP’s “verConfrontos” e “visualizarCalendario”. O caso de uso de cada página JSP foi descrita anteriormente na descrição da *Action*. Estas páginas irão conter o código *html* que será preenchido com os dados do *Model*. Este componente corresponde ao componente da view do padrão *Model View Controller (MVC)*.

Model: Cada modelo representa uma abstração para os dados de cada operação na interface gráfica. Cada operação tem um modelo com atributos que depois são atribuídos em runtime na página JSP quando o utilizador vai executando e inserindo os vários dados na interface *web*. Isto corresponde ao componente do *Model* do padrão *Model View Controller (MVC)*. Para efeitos de abstração e simplificação do diagrama criámos um componente “Model” que contém todos os *Models* utilizados no sistema. No sistema existem os seguintes modelos: “RegistrarResultadoModel”, “VerConfrontosModel”, “VisualizarCalendarioModel” e “VisualizarTorneiosModel”, cada um destes é utilizado em conjunto com a respetiva página JSP para atribuir os atributos que depois são utilizados na *action* e passados para o *business* do TornGest.

index.html: Este componente representa a página inicial da interface gráfica, a primeira página que o utilizador vê quando lança a aplicação *web* e que contém ligações para as duas funcionalidades.

TorneioServicesFacade: Esta interface é um *bean* de sessão sem estado e oferece os métodos necessários ao caso de uso de registar o resultado de um encontro. O *business* do TornGest contém uma classe “TorneioService” que implementa esta interface. Esta

técnica permite à aplicação *web* não saber detalhes da implementação dos vários objetos no *business* uma vez que neste *bean* apenas são utilizados dados de tipos primitivos e também ajuda na ligação enviando objetos menos pesados, mas em maior número. Para efetuar estas operações este *bean* liga-se com a aplicação *web* através de chamada Java e através de uma chamada *SOAP* para a camada de negócio do TornGest.

CalendarioServiceFacade: Esta interface é um *bean* de sessão sem estado que oferece os métodos necessários ao caso de uso de visualizar o calendário. O business do TornGest contém uma classe “CalendarioService” que implementa esta interface. A lógica por detrás desta fachada do serviço é a mesma que foi descrita anteriormente no *TorneioServicesFacade*.

TornGestBusiness: Este é o componente correspondente à camada de negócio do TornGest. Este componente é uma aplicação desenvolvida na framework *Java EE* fala com a aplicação *web* através de chamadas *SOAP* aos *beans* de sessão sem estado referidos anteriormente. Para uma vista mais detalhada, é favor consultar a vista de Componentes e Conectores do *TornGestBusiness*.

Os diagramas de atividade UML seguintes mostram ambos os casos de uso que são realizados pela *WebApp*, o de registrar um resultado e verificar o calendário de um jogador. (Para simplificação estes diagramas não mostram a interação com a base de dados.)

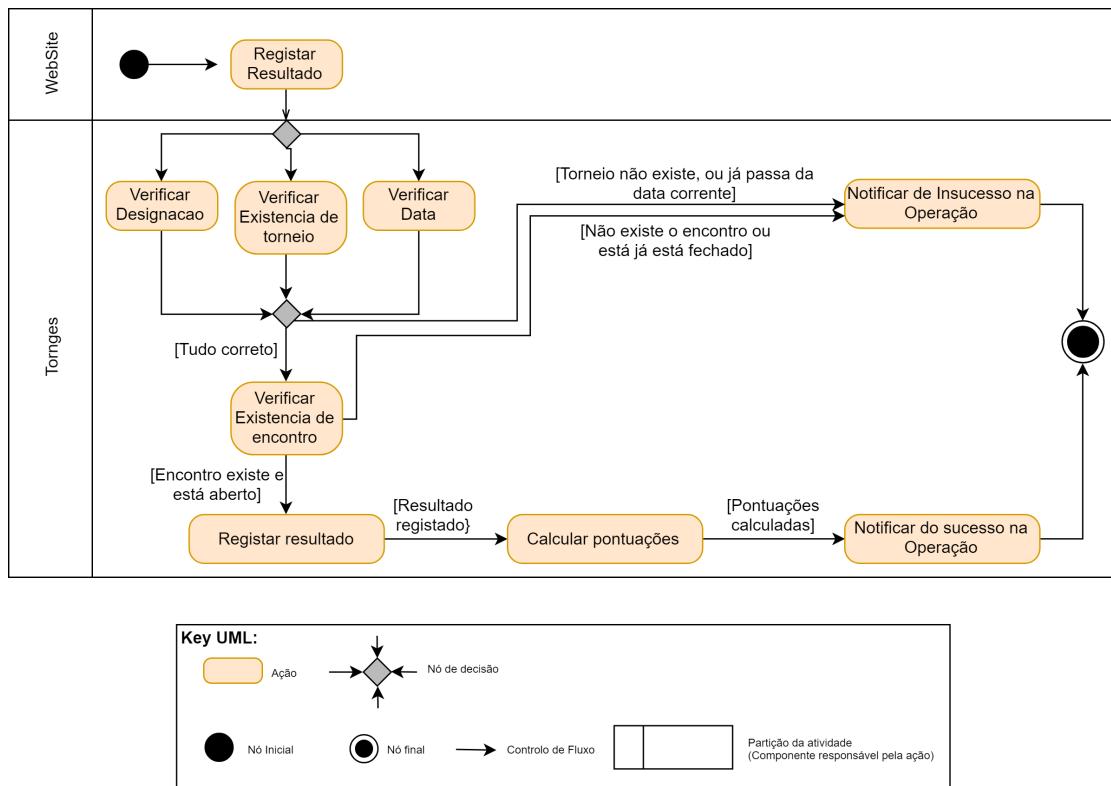


Figura 3.10: Vista de Interação Top Level SOA do RegistarResultado

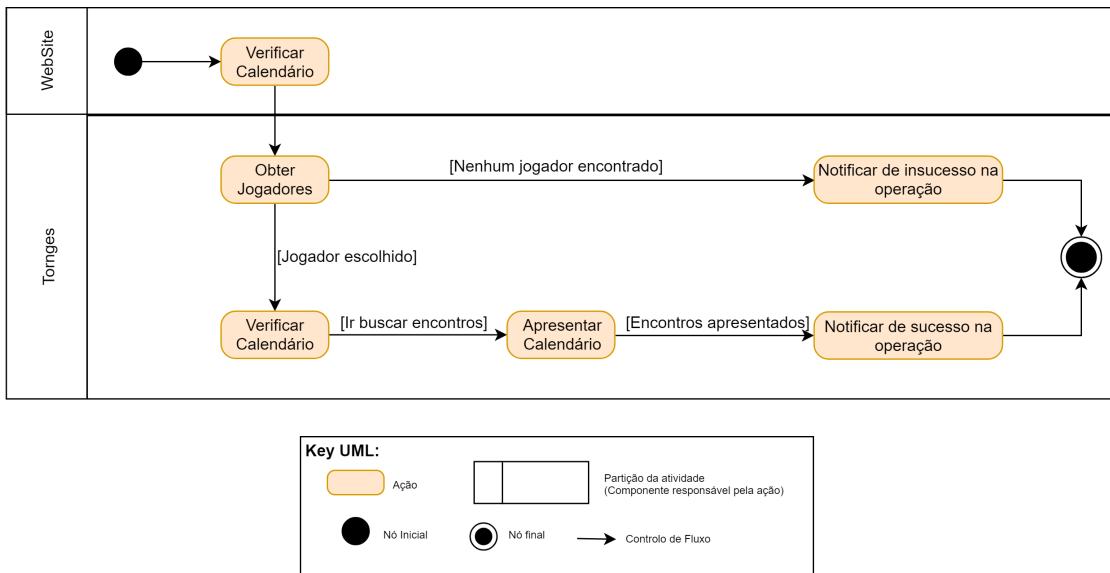


Figura 3.11: Vista de Interação Top Level SOA do VerificarCalendario

TornGestDB: Este componente representa uma base de dados relacional *MySQL* que armazena dados sobre jogadores, modalidades, equipas, torneios, encontros e confrontos. Para uma vista mais detalhada sobre as várias entidades nesta base de dados e os vários atributos é favor consultar a vista do modelo de dados.

Diagrama de Contexto: N/A.

Mecanismos de Variabilidade: N/A.

Racionalização:

A framework **Java EE** foi assim escolhida para desenvolver este sistema pois foi a **framework** lecionada em Construção de Sistemas de Software, unidade curricular onde este projeto foi desenvolvido, também porque apresenta abstrações bastante boas para componentes como os *beans* de sessão e *Servlets* e facilita também o desenvolvimento desta aplicação web através do padrão *Model View Controller* tendo um *Front Controller* que recebe todos os pedidos do utilizador, redirecionando-os para as várias ações. Cada ação vai carregar as páginas JSP na interface. Existem também modelos que são preenchidos através do preenchimento dos campos nas páginas JSP e que são utilizados na action para capturar os dados e enviá-los para o resto da computação do caso de uso.

3.1.2.4 Vista de Componentes e Conectores do GUI

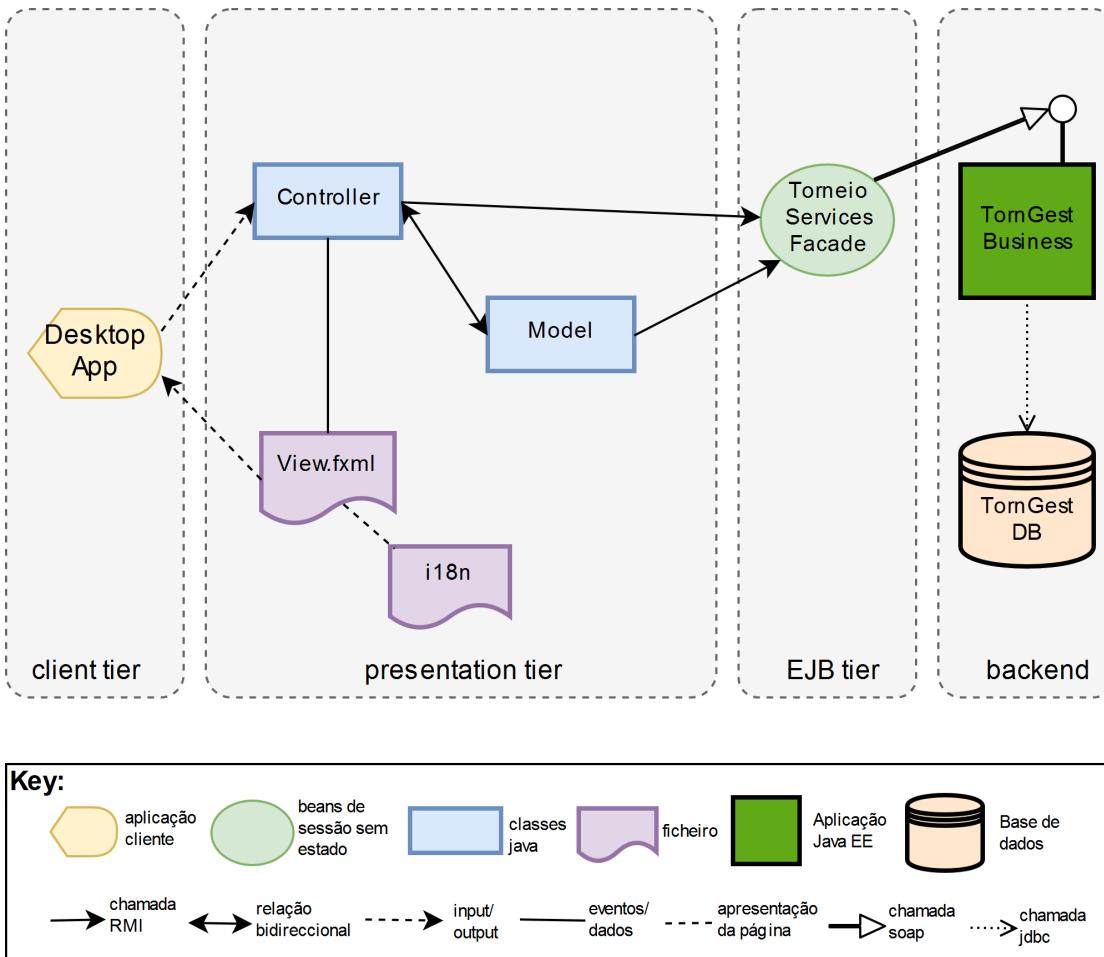


Figura 3.12: Vista de Componentes e Conectores do GUI

Descrição da Vista:

Esta vista descreve uma arquitetura cliente-servidor em 4 camadas, sendo esta uma representação de como a aplicação *desktop* comunica entre si e com o *business* da aplicação.

Background Arquitetural:

Para o caso do componente da GUI do TornGest foi utilizado como padrão principal de organização o padrão *MVC* ou *Model View Controller* e isso reflete-se na organização das unidades de código. Mais especificamente foi utilizado o padrão *Page Controller*, existindo uma classe *Controller* por “página” de apresentação. Associados a estes *controllers* estão os models que guardam valores que serão usadas para apresentação no ecrã ou valores introduzidos pelo utilizador. Finalmente existe um ficheiro *FXML* que representa a parte estática do programa, ou seja, um ecrã que o utilizador vê. A utilização do padrão *MVC* tem impacto no projeto na medida que há uma *Separation of Concerns* da interface do resto da lógica e dados do sistema.

Catálogo de Elementos:

Desktop App: Esta é a aplicação desktop com a qual o cliente interage. A aplicação arranca com a classe *Startup* que guarda o *bean* e passa aos controllers para usarem nas

susas operações.

view.fxml: A view é a apresentação da página, apresentação esta que é guardada num ficheiro fxml. Este ficheiro foi feito com uso do SceneBuilder. É aqui que é apresentado o output ao cliente, sendo que o tratamento dos dados introduzidos aqui é tratado no controller. Existem várias views para o TornGest, menuPrincipal, criarTorneio e registarParticipante.

i18n: No i18n estão guardadas as várias definições das linguagens, sendo as opções português e inglês.

Controller: No controller é tratado o input obtido na view, e guardado no model respectivo através de uma relação de bidireccionalidade. O controller chama os vários métodos do business através de chamadas Java RMI, usando um bean de sessão sem estado passado a ele no *Startup*. Depois de chamar as funções que necessita apresenta um mensagem de sucesso ou insucesso no ecrã. Existem quatro controllers, o baseController que oferece mensagens de erro e informação e três outros que extendem este. O MenuPrincipalController apenas redirecciona para os outros dois, o CriarTorneioController e o RegistarParticipanteController.

Model: No model são guardadas as informações necessárias, e são também buscadas ao business listas de elementos para apresentar ao utilizador. Faz isto da mesma maneira que o controller, através de uma chamada Java RMI ao serviço oferecido. Existem dois models, o CriarTorneioModel e o RegistarParticipanteModel.

- **Torneio Services Facade:** Este beans não guarda estado e corresponde à interface ITorneioServiceRemote oferecida pelo business. A aplicação GUI obtém o bean através de injecção de dependências.
- **TornGest Business:** É aqui que é tratada toda a lógica da aplicação, oferecendo as seguintes interfaces: *ICalendarServiceRemote* e *ITorneioServiceRemote*.

Para a GUI apenas uma das interfaces remotas é chamada, e esta é chamada é feita através de Java RMI. As duas operações que o GUI trata são a criação de um torneio e o registo de um jogador num torneio.

Os diagramas UML seguintes mostram a ordem de operação das duas funcionalidades, sendo a interacção com a base de dados removida para simplificação.

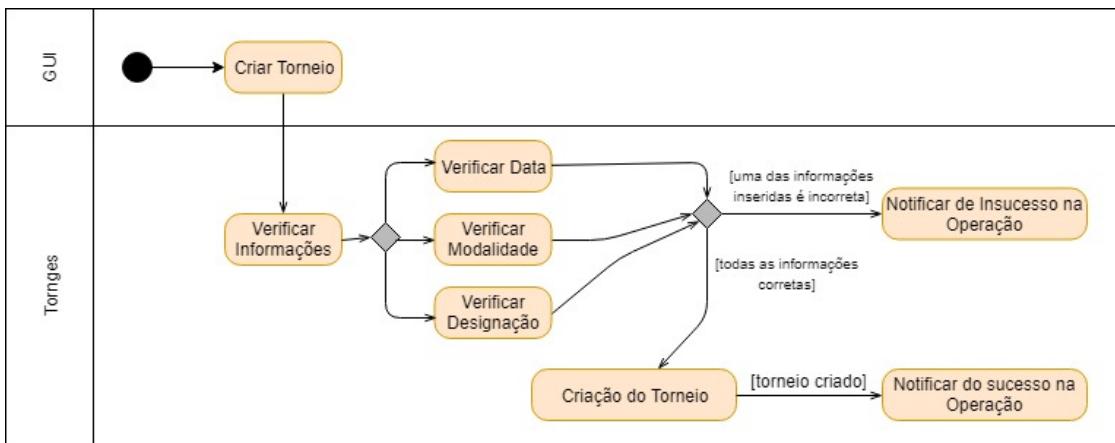


Figura 3.13: Vista de Interação Top Level SOA da Criação de Torneio

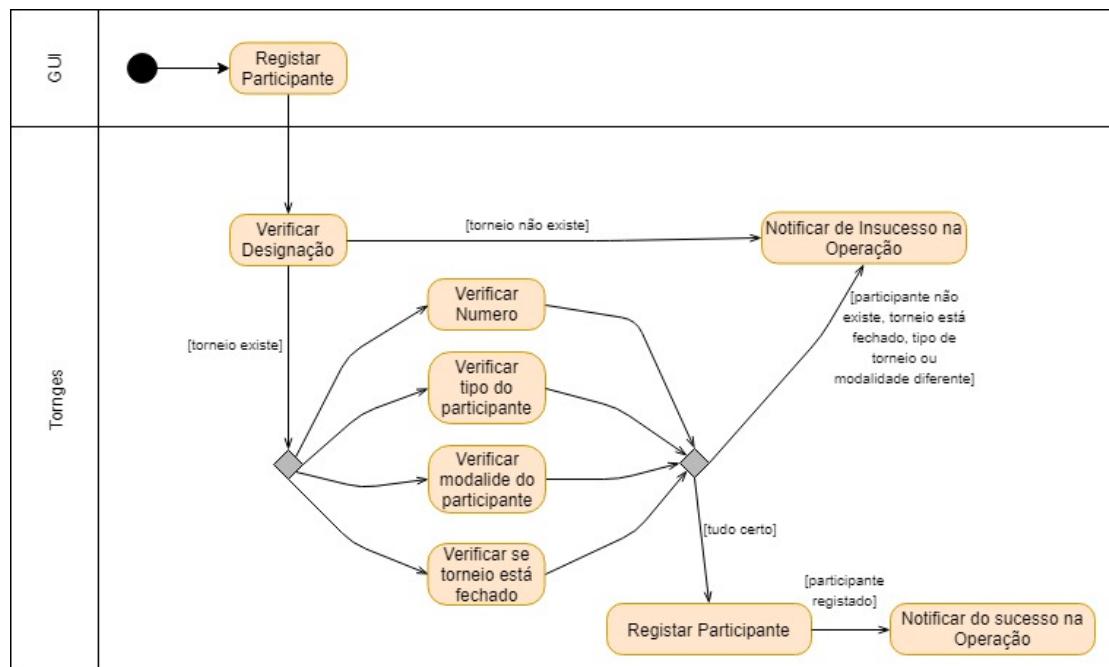


Figura 3.14: Vista de Interação Top Level SOA do RegistarParticipante

TornGest DB:

Esta é a base de dados com a qual o business comunica e realiza operações sobre os torneios e participantes.

3.1.3 Vista de Alocação

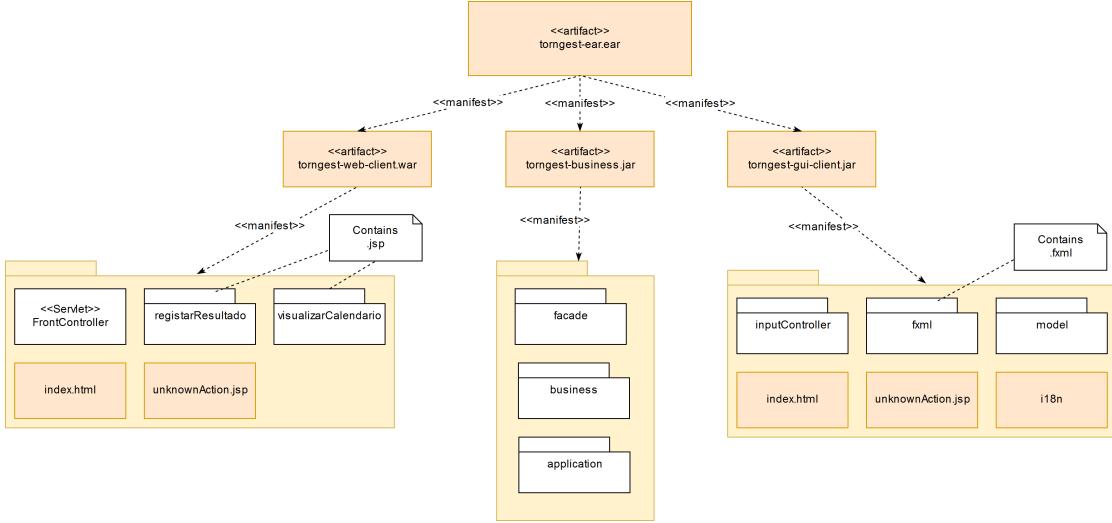


Figura 3.15: Vista de Alocação

Descrição da vista:

Esta vista de instalação apresenta os três diferentes elementos e as suas decomposições do *torngest-ear.ear*. Este encontra-se subdividido em três principais elementos que correspondem a pontos essenciais da nossa aplicação.

Catálogo dos Elementos:

torngest-gui-client.jar: Este *jar* foi gerado contendo todos os elementos necessários para que seja possível apresentação de um cliente *GUI* e o seu acesso aos *Web Services* que permitem interagir com o sistema. Encontram-se descritos nas vistas anteriores a decomposição e os componentes e conectores existentes neste módulo. Este módulo é instalado na máquina do cliente onde é necessário se encontrar instalado o *Java* de modo a poder correr o programa.

torngest-web-client.war: Com semelhança ao *torngest-gui-client.jar* este pacote apresenta os elementos necessários para correr uma aplicação *web* acedendo aos *Web Services* através de um *Front Controller*. Neste caso tudo o que é necessário é que o cliente tenha um *Web Browser* para poder aceder à aplicação e interagir com o sistema. As vistas de decomposição e componentes e conectores anteriormente apresentadas descrevem a modularização e o comportamento apresentado por este módulo.

torngest-business.jar: Esta representa a parte principal da nossa aplicação. Este *jar* é instalado e executado numa máquina servidora como o *WildFly*. Este apresenta todos os serviços que podem ser acedidos por aplicações externas e realiza a ligação a uma base de dados para persistir as alterações realizadas. As vistas de decomposição e componentes e conectores anteriormente apresentadas descrevem a modularização e o comportamento apresentado por este módulo.

3.2 Mapeamento entre Vistas

<i>Vista Top Level SOA</i>	<i>Vista de Alocação</i>
componente: torngest-business.jar	artefacto: TornGes
componente: torngest-gui-client.jar	artefacto: GUI Cliente
componente: torngest-web-client.war	artefacto: Website Consumidor

Tabela 3.1: Mapeamento entre a Top Level Soa e Vista de Alocação

<i>Vista de Multi-tier do GUI</i>	<i>Vista de Alocação</i>
componente: torngest-business.jar	artefacto: TornGest Business
componente: Controller	módulo: inputController
componente: Todos os <i>Model</i>	módulo: model
componente: View.fxml	módulo: fxml
componente: i18n	módulo: i18n

Tabela 3.2: Mapeamento entre a Multi-Tier do GUI e Vista de Alocação

<i>Vista de Multi-tier do Web</i>	<i>Vista de Alocação</i>
componente: torngest-business.jar	artefacto: TornGest Business
componente: FrontController	servlet: FrontController
componente: Página JSP, unknownAction.jsp	módulo: registrarResultado, visualizarCalendario
componente: index.html	ficheiro: index.html
componente: TorneioServiceFacade	módulo: facade
componente: CalendarioServiceFacade	módulo: facade

Tabela 3.3: Mapeamento entre a Multi-Tier do Web e Vista de Alocação

<i>Vista de Decomp. do Business</i>	<i>Vista de C&C do Business</i>
pacote: application	componentes: CalendarioService, TorneioService
pacote: handler	componentes: VisualizarCalendarioHandler, RegistrarParticipanteHandler, CriarTorneioHandler
pacote: business	componentes: Todos os <i>Beans</i> representados, <i>Entity Beans</i> e <i>Stateless Session Beans</i>

Tabela 3.4: Mapeamento entre a Vista de Decomp. e C&C do Business

<i>Vista de Decomp. do Web</i>	<i>Vista de Multi-tier do Web</i>
pacote: actions	componentes: Action
pacote: webapp	componentes: Página jsp, unknownAction.jsp
pacote: model	componentes: Todos os <i>Models</i> apresentados
pacote: inputController	componente: FrontController
pacote: resources	componente: app.properties

Tabela 3.5: Mapeamento entre a Vista de Decomp. e Multi-tier do Web

<i>Vista de Decomp. do GUI</i>	<i>Vista de Multi-tier do GUI</i>
pacote: inputController	componente: Controller
pacote: client	componentes: Desktop App
pacote: i18n	componente: i18n
pacote: model	componentes: Todos os <i>Model</i> apresentados

Tabela 3.6: Mapeamento entre a Vista de Decomp. e Multi-tier do GUI

Capítulo 4

Bibliografia

Addison-Wesley. (2011). *Documentating Software Architectures*. Addison-Wesley.

Baghwant. (8 de Maio de 2013). *Software Architecture Guidelines*. Obtido de Technowiki: <https://technowiki.wordpress.com/2013/05/08/software-architecture-document-guidelines/>

Google. Obtido de *Software Architecture in Practice - Implementation View*: <https://sites.google.com/site/softwarearchitectureinpractice/9-documenting-software-architecture/d-allocation-views/b-implementation-view>

Lopes, A. (2018-19). *Design de Software, Arquitetura de Software*. Faculdade de Ciências da Universidade de Lisboa, Departamento de Informática.

P. M., & D. S. (09 de Julho de 2009). Obtido de *Adventure Builder - Software Architecture Document (SAD)*: https://wiki.sei.cmu.edu/sad/index.php/The_Adventure_Builder_SAD

Software, C. d. (2018). *Projeto: 1^a parte*. Faculdade de Ciências da Universidade de Lisboa.

Software, C. d. (2018). *Projeto: 2^a parte*. Faculdade de Ciências da Universidade de Lisboa.

Tennage, P. (08 de 04 de 2016). *Sample Software Architecture Document*, 1.0. Obtido de LinkedIn SlideShare: <https://pt.slideshare.net/PasinduTennage/sample-software-architecture-document>

W. G., Y. L., X. B., & Z. S. (2010). *Basic Contents of Software Architecture Design*. (p. 5). Taiyuan, China: IEEE.
Obtido de <https://ieeexplore.ieee.org/document/5622990>