Software Document Architeture (SAD)
Design Software

André Nicolau 47880
Ana Sofia 47297
Gonçalo Neto 55110

November 24, 2019

Contents

1 Purpose and Scope of the SAD

No documento presente será descrita a arquitetura de software do sistema de gestão de torneios TornGest, sendo apresentados os elementos principais de software deste, as propriedades externas destes e as relações entre estes, com o objetivo de compreender as decisões feitas no design deste sistema.

Na secção seguinte, serão descritos os stakeholders do sistema, sendo esta seguida pela secção 3, onde será posto o foco nos detalhes do sistema e dos seus objetivos, como também os requisitos de qualidade que advêm das necessidades dos stakeholders e inerentes ao sistema em questão. Seguidamente, na secção 4, serão documentadas as Vistas do sistema que achámos pertinente documentar, e na secção seguinte, as relações entre estas. Por fim, na secção 6, apresentamos os resultados da avaliação do sistema com uso da ferramenta Sonarqube, retirando conclusões sobre como as decisões de desenho feitas afetam o sistema nos seus vários aspetos.

2 Stakeholders

Este sistema tem diversos stakeholders, cada um com as suas prioridades em relação ao sistema e às suas qualidades.

Stakeholder	**Tarefas**	**Necessidades**
Equipa de desenvolvimento	Desenvolvimento e manutenção do sistema	Desenvolvimento e manutenção rápidos e fáceis
Empresa da equipa de desenvolvimento	Comercialização do sistema	Custo de desenvolvimento reduzido e manutenção facilitada
Gestores de torneio	Utilização do sistema para gerir torneios	Fácil aprendizagem e boa fiabilidade de sistema
Associações de desporto	Obtenção e instalação do sistema	Custo de compra reduzido e instalação simples

Figure 1: Stakeholders

3 Architecture Background

3.1 Problem Background

3.1.1 System Overview, Goals and Context

System Overview

O TornGest é um sistema de gestão de torneios, baseado no software Desporgest de gestão de provas desportivas. Com este sistema, é permitida a criação de torneios, como também a manutenção destes, de acordo com o sistema de Elo.
Cada torneio é composto por um conjunto de várias provas desportivas entre 2

participantes, sendo considerados tipos de torneios individuais ou de equipa.

System Overview
Os objetivos deste sistema são permitir aos seus utilizadores, mais especificamente, aos gestores de torneios que utilizarão esta ferramenta, através de um cliente web, o registo de torneios e seus participantes, e através de um cliente desktop, a classificação dos torneios e visualização do calendário dos jogadores participantes.

Context

- Diagrama de Casos de Uso

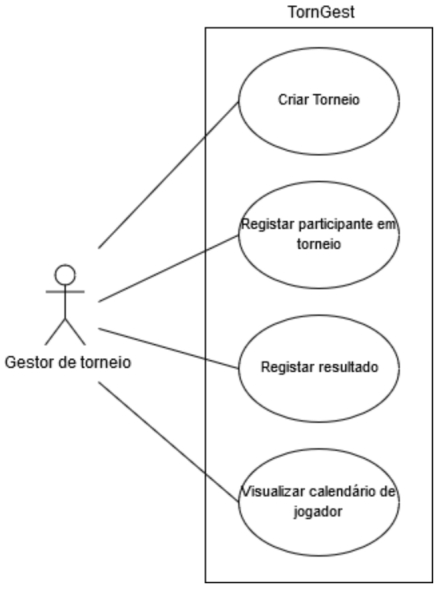

Figure 2: casos de uso

- Casos de uso detalhado

  Caso de uso 1: Criar torneio - Gestor inicia criação novo torneio. Sistema indica modalidades existentes, pede modalidade, designação do torneio, tipo do torneio, nº participantes, nº encontros entre pares de oponentes, data de início de torneio e duração em dias. Gestor fornece esta informação e sistema valida-a, criando um novo torneio aberto a registo de participantes.

  Caso de uso 2: Registar participante em torneio - Gestor inicia registo de X (nº de inscrição) participante (jogador/equipa) em X (designação) torneio. Sistema valida informação recebida, registando participante no torneio e, no caso de ter sido atingido o nº limite de participantes

neste, fecha o registo neste e cria todos os seus encontros.

Caso de uso 3: Registar resultado de encontro - Gestor inicia registo de resultado de encontro de torneio X. Sistema valida informação recebida e mostra encontros do torneio sem resultados (nº de encontro, nomes do 1º e 2º oponente) ordenados por nº de encontro. Gestor escolhe o nº de um encontro e indica o resultado entre 1 dos 3: vitória, empate ou derrota (do 1º oponente). Sistema mais uma vez verifica se os dados inseridos são válidos em contexto, regista o resultado, atualiza as pontuações dos jogadores intervenientes e das suas equipas (no caso de se tratar de torneio de equipa). Sistema continuará a repetir este processo até gestor especificar que quer sair.

Caso de uso 4: Visualizar Calendário de Jogador - Gestor inicia visualização de calendário de confrontos do jogador, fornecendo nº de inscrição e data. Sistema valida pedido e mostra lista ordenada por data com confrontos do jogador (equipa e individuais) em torneios posteriores à data atual. Entradas da lista contêm: designação de torneio, data de início, jogador adversário e diferencial de pontos, e nº de encontros.

3.1.2 Quality Requirements

Modificabilidade
Medida relacionada com o esforço na realização de eventuais mudanças e manutenção ao sistema.

Cenário:	Alteração de tamanho de letra na interface
Objetivos:	Alteração rápida
RQ relevantes:	Modificabilidade
Estímulo:	Pedido de alteração
Fonte do estímulo:	Gestor de projeto
Ambiente:	Design Time
Artefacto:	Letra da interface
Resposta:	Efetuar alteração
Medida de resposta:	Demorar menos de 1 hora

Figure 3: Modificabilidade

Fiabilidade
Medida da garantia de que o sistema executa as suas funções sem falhas inesperadas, num determinado período de tempo.

Portabilidade
Medida da agilidade de execução do sistema, de forma eficiente, em diferentes

Cenário:	Registo de resultado de encontro
Objetivos:	Registo rápido e sem falhas
RQ relevantes:	Fiabilidade
Estímulo:	Pedido de registo
Fonte do estímulo:	Gestor de torneio
Ambiente:	Run Time
Artefacto:	Torneio
Resposta:	Efetuar registo de resultado
Medida de resposta:	Demorar menos de 5 segundos; sem falhas

Figure 4: Fiabilidade

ambientes, por exemplo, diferentes arquiteturas (tanto software como hardware) ou linguagens de programação. (Alteração do SGBD, por exemplo)

Cenário:	Alteração de SGBD
Objetivos:	Alteração simples
RQ relevantes:	Portabilidade
Estímulo:	Pedido de alteração
Fonte do estímulo:	Gestor de projeto
Ambiente:	Design Time
Artefacto:	SGBD
Resposta:	Efetuar alteração
Medida de resposta:	Demorar menos de 3 dias; alteração sem sequelas em outros módulos

Figure 5: Portabilidade

Usabilidade
Grau da dificuldade de aprendizagem e utilização da interface, e por conseguinte, produtividade que provém da utilização.

Testabilidade
Propriedade relacionada com a facilidade de demonstração que o sistema opera da maneira pretendida.

Custo
Propriedade referente aos recursos necessários para produção e manutenção do sistema versus lucro com a produção do mesmo.

Cenário:	Visualizar calendário de jogador
Objetivos:	Pedido de visualização bem sucedido
RQ relevantes:	Usabilidade
Estímulo:	Pedido para visualizar calendário de jogador
Fonte do estímulo:	Gestor de torneio
Ambiente:	Run Time
Artefacto:	Calendário de jogador
Resposta:	Demonstração de calendário
Medida de resposta:	Demorar menos de 5 minutos; Máximo de 10 cliques; Máximo de 3 erros por parte do utilizador

Figure 6: Usabilidade

Cenário:	Testar criação de torneio
Objetivos:	Encontrar falhas no sistema com facilidade
RQ relevantes:	Testabilidade
Estímulo:	Pedido de teste
Fonte do estímulo:	Gestor de projeto
Ambiente:	Build Time
Artefacto:	Torneio
Resposta:	Demonstração de resultados de teste
Medida de resposta:	Nível de detalhe dos testes, facilidade de execução de bateria de testes e verificação do estado do sistema

Figure 7: Testabilidade

3.2 Solution Background

Neste ponto vamos descrever a arquitetura usada neste projeto, a razão pela qual a mesma foi escolhida para o contexto do projeto e os requisitos que estão a ser cumpridos pela arquitetura.

3.2.1 Architectural Approaches

Este projeto foi dividido em 4 camadas essenciais e são elas: a camada de apresentação, camada de serviços, camada do negócio e a camada de acesso aos dados. Está organizada da seguinte forma:

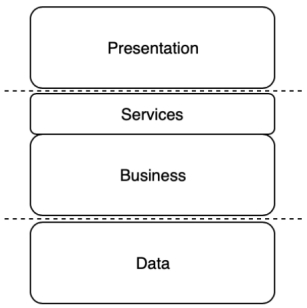

Figure 8: Layers

Esta arquitetura em camadas (Layered Pattern) parte do princípio de que as camadas superiores usam as imediatamente inferiores, isto é, a camada de negócio não pode usar a camada de apresentação e a camada de apresentação não pode usar a camada dos dados.
A camada de apresentação trata da forma em que os utilizadores irão interagir com a aplicação. A camada dos serviços é como uma interface para a camada de apresentação usar para ter "acesso" à camada de negócio. A camada de negócio contém toda a lógica do negócio da aplicação. A camada dos dados é onde se trata do acesso aos dados em bases de dados(na maioria dos casos).
Este projeto suporta a interação com o utilizador pela web e localmente(recorrendo ao JavaFX). Para ambos, é usada a abordagem do MVC (Model-View-Controller) que é um exemplo de arquitetura tendo em vista a camada de apresentação. O esqueleto da aplicação web tira partido do Java EE Web Container, enquanto que a GUI (Graphical User Interface) acede à camada de negócio utilizando um RMI (Remote Method Invocation), recorrendo ao Java EE Application Client Container.
Na camada do negócio é onde estão definidas as características do negócio, fornecendo meta-dados para persistir objetos ao JPA. Nesta camada está a ser usado o Domain Model como estrutura do código, e é também autonomizada para poder ser acedida por qualquer tipo de cliente, com Java EE EJB Container.
Por fim, a camada de dados é onde se processam as queries vindas da camada de negócio, com uso de JPA, seguindo o padrão Data Mapper.

3.2.2 Requirements Coverage

A arquitetura implementada neste sistema consegue cobrir os vários requisitos.

A nível da Modificabilidade devido ao uso do Domain Model na camada do negócio. Uma das propriedades deste modelo é a facilidade de lidar com alterações no sistema.

Falando em Portabilidade, a arquitetura em camadas tem essa vantagem. Por exemplo a camada dos Dados pode mudar mas a camada de cima, neste caso a camada do negócio não ir necessitar de mudança.

Em relação à Fiabilidade existem alguns problemas no fazer as tarefas que é suposto fazer. Irá ser feita uma análise mais aprofundada em relação a bugs na secção mais abaixo do Sonaqube

A nível de Usabilidade, as interfaces gráficas para uso dos utilizadores não são exigentes.

Relativamente à Testabilidade, como a aplicação é feita recorrendo ao JavaEE e isso traz-nos uma visão mais clarificada de várias vertentes da aplicação a nível de teste.

Por fim, a aplicação não requer nenhum tipo de hardware extra. Corre na mesma máquina. Só necessita da instalação prévia do Java Enterprise Edition e do Wildfly 10.0.0/1 da JBoss.

4 Views

Na secção presente iremos apresentar as diferentes vistas, isto é, as diferentes abstrações do sistema que achámos relevante documentar, de acordo com os requisitos de qualidade documentados.

4.1 Module Views

4.1.1 Decomposition View

Aqui documentamos as vistas de módulos, que representam as estruturas de código e dados do sistema. Estas vistas servem de apoio à manutenção e evolução do sistema, de modo que achámos relevante estas serem documentadas, já que é posta ênfase na modificabilidade do sistema.

4.1.1.1 View Description

Nesta vista o sistema TornGest é decomposto nas suas unidades de implementação, em específico, os módulos e submódulos nos quais estão divididos as responsabilidades do sistema.

4.1.1.2 View Packet Overview

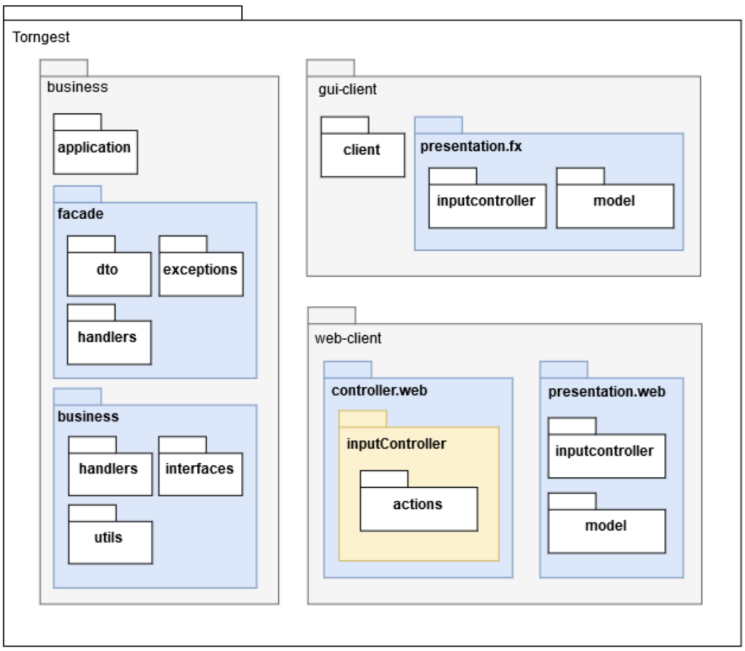

Figure 9: Decomposição

4.1.1.3 Architecture Background

business Contém os pacotes associados à lógica de negócio do sistema.

business.handlers - Este pacote contém as classes que representam os handlers dos 4 casos de uso, visualização de calendário de jogador, criação de torneio, registo de participantes e registo de resultados de encontro.

business.interfaces - Pacote com as classes que implementam as interfaces com operações específicas a cada um dos casos de uso.

business.utils - Contém a classe auxiliar responsável pelo cálculo do Elo, utilizado na classificação dos jogadores.

application - Pacote que contém as classes CalendarioService, que implementa os serviços remotos relacionados com a visualização do calendário de jogador, e TorneioService, que implementa os serviços remotos relacionados com os 3 casos de uso restantes.

facade - Encapsula os pacotes dto, exceptions e handlers, escondendo a complexidade do seu subsistema de pacotes e lidando com a mesma, de modo a se poder alterar o subsistema sem afetar os cliente.

facade.dto - Contém os DTOs (Data Transfer Objects) utilizados para car-

regar os dados entre a camada de negócio e dos clientes. Deste modo, objetos da camada de negócio ou de acessos a dados não são acedidos pela parte responsável pela interação com o utilizador.

facade.exceptions - Contém a classe que representa a exceção de mais alto nível no sistema, utilizada para encapsular outras exceções de menor nível, na camada de negócio. Quaisquer das exceções em níveis inferiores devem estender essa mesma classe.

facade.handlers - Contém as classes que implementam as interfaces gerais utilizadas pelos serviços remotos para os casos de uso já mencionados.

gui-client - Lida com o input do gui-client e acede à camada de negócio, com uso do padrão Model-View-Controller.

client - Contém classe Main, aplicação cliente para utilização dos serviços dos 2 casos de uso para o cliente desktop.

presentation.fx - Contém pacotes inputController e model, que lidam com o input do gui-client e transferem os pedidos para a camada de negócio.

presentation.fx.inputController - Contém as classes que manipulam os dados inseridos e preparam a resposta ao input, enviando este ao Model, para serem processadas, e para a View, onde serão realizadas as operações.

presentation.fx.model - Conecta as camadas View e Controller, contém as classes com os dados da aplicação, regras, lógica e funções. Permite o acesso aos dados para operações.

web-client - Lida com o input do web-client e acede à camada de negócio, com uso do padrão Model-View-Controller.

controller.web.inputController - Contém as classes handlers, que processam as ações para os pedidos HTTP.

presentation.web - Contém pacotes inputController e model, que lidam com o input do gui-client e transferem os pedidos para a camada de negócio.

presentation.web.inputController - Contém as classes que manipulam os dados inseridos e preparam a resposta ao input, enviando este ao Model, para serem processadas, e para a View, onde serão realizadas as operações.

controller.web.model - Conecta as camadas View e Controller, contém as classes com os dados da aplicação, regras, lógica e funções. Permite o acesso aos dados para operações.

4.1.2 Uses View

4.1.2.1 View Description

Esta view descreve as dependências de cada package na aplicação entre si e com packages exteriores à mesma.

4.1.2.2 View Packet Overview

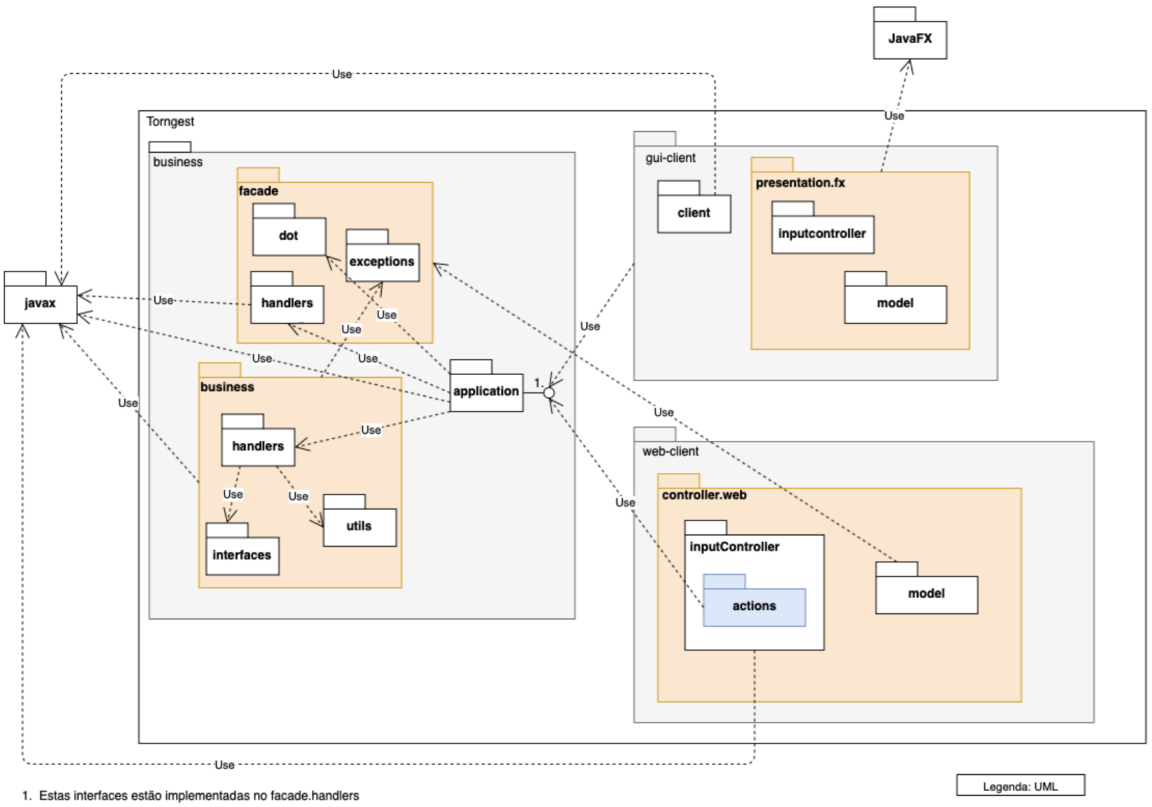

Figure 10: Uses

4.1.2.3 Architecture Background

Nesta vista podemos identificar que muitos módulos dependem do pacote javax(nomeadamente facade.handlers, business, gui-client.client e web-client.controller.web.inputController.ac e na vertente da apresentação o pacote gui-client.presentation.fx depende do pacote externo JavaFX.

4.1.3 Context Diagram

4.1.3.1 View Description

O diagrama relativo ao contexto demonstra o funcionamento da interação entre utilizador e sistema e como os casos de uso se dividem, por gui-client e web-client, numa vista de alto nível.

4.1.3.2 View Packet Overview

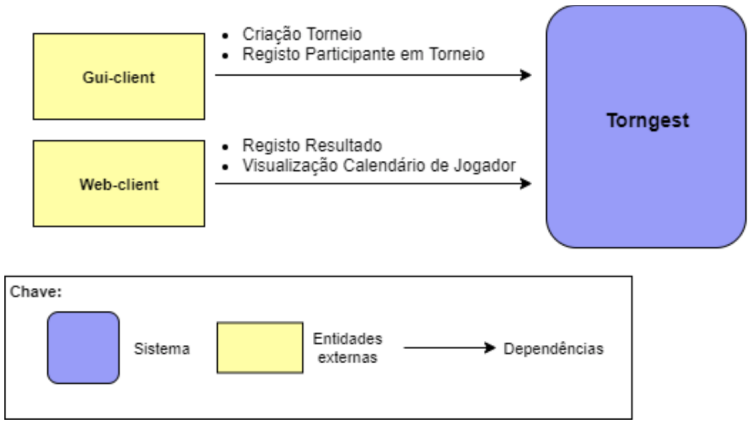

Figure 11: Context

4.1.3.3 Architecture Background

O sistema fornece ao utilizador (gestor de torneio) a possibilidade criar torneios e registar nestes os respetivos participantes através de uma aplicação web, como também lhe permite registar os resultados dos encontros nestes torneios e ainda visualizar o calendário dos jogadores através de uma interface gráfica no desktop.
O sistema TornGest processa os pedidos e prepara uma resposta ao utilizador, acedendo a uma base de dados onde adquire ou altera os dados necessários, de modo a responder ao pedido respetivo.

4.1.4 Data Model

4.1.4.1 View Description

Nesta vista, é apresentada a organização da base de dados do sistema nas suas respetivas entidades (tabelas), com as suas relações e propriedades.

4.1.4.2 View Packet Overview

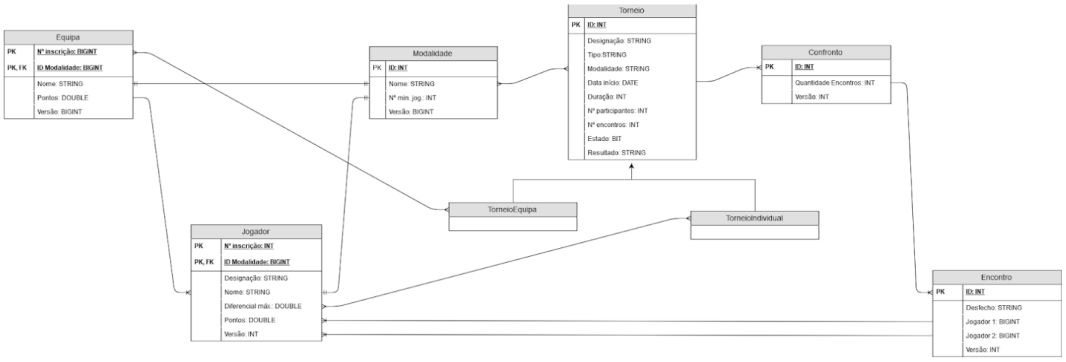

Figure 12: Data

4.1.4.3 Architecture Background

O sistema de gestão de base de dados do TornGest utiliza MySQL. Deste modo, todos os dados que correspondam a sequências de números Long são armazenados na BD como BIGINT, os objetos de formato Calendar, como DATE, e objetos Boolean no formato BIT (1 ou 0). Como não é possível o armazenamento de objetos em sistemas de armazenamento de dados, estes são armazenados nas tabelas por identificadores, em formato BIGINT (Jogador 1 e Jogador 2 na tabela Encontro).

As entidades (tabelas) Equipa e Jogador possuem Foreign Keys com o identificador da modalidade a que pertence cada equipa ou jogador, que relaciona as tabelas correspondentes à de Modalidade. Isto garante a integridade referencial dos dados nesta coluna em ambas as entidades. Para além disto, a utilização de Primary Keys e Foreign Keys beneficia o desempenho (indiretamente) já que são automaticamente indexadas para otimização de procura, sendo que as referências FK serão essenciais para 1 caso de uso do sistema (Registar participante em torneio).

Coluna **Versão** - Utilizada pelo JPA de modo a que sejam evitadas falhas devido a estados inconsistentes de dados na BD. Com isto, a API tem a capacidade de detetar tentativas de modificação ao mesmo dado simultaneamente, enviando uma exception à final de ambas as transações.

4.2 Components and Connectors

4.2.1 Components and Connectors View

4.2.1.1 View Description

Esta vista vai-nos transmitir como é que os diferentes componentes do sistema

estão conectados entre si e de que forma.

4.2.1.2 View Packet Overview

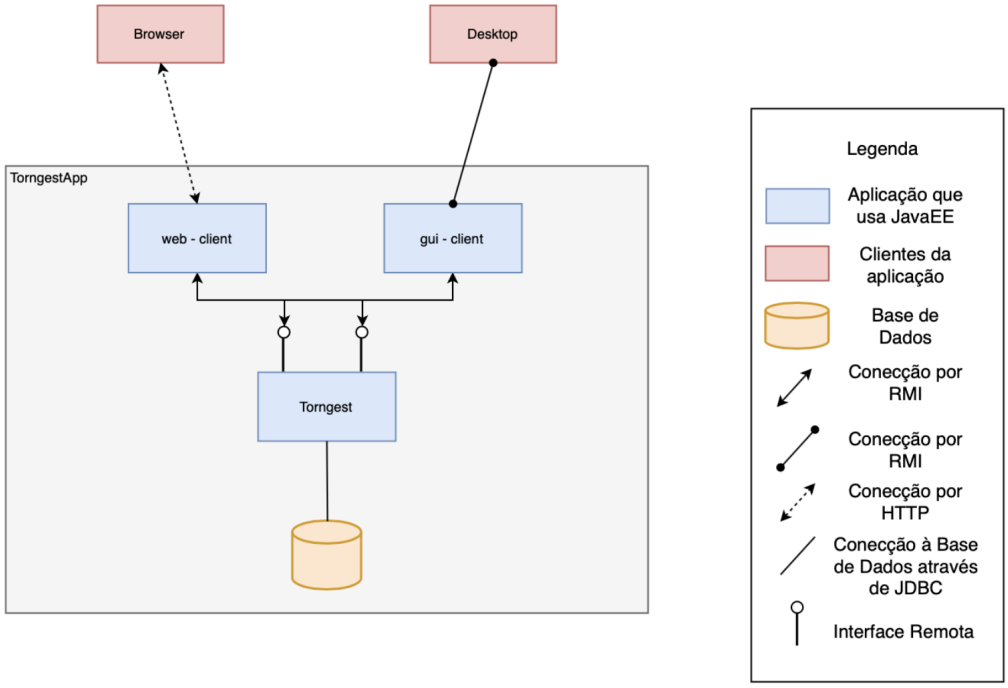

Figure 13: Componentes e Conectores

4.2.1.3 Architecture Background

web-client - Aplicação implementada com o Java EE que consiste num conjunto de ficheiros JSP e HTML. Trata-se da parte do sistema TornGest à qual os clientes acedem através de um browser.
São processados os pedidos HTTP vindos dos utilizadores que pretendem aceder/registar nas diversas partes do sistema.

gui-client - Aplicação implementada com o Java EE, recorrendo ao JavaFX. Pode ser executada no computador do utilizador, permitindo-o criar provas desportivas.

torngest - Aplicação implementada com o Java EE que consiste em receber pedidos do web-client(via HTTP) ou gui-client(via RMI) e enviar uma resposta aos mesmos pelos mesmos meios.

4.2.2 Multi-Tier View

4.2.2.1 View Description

Nesta vista são apresentados os tiers da aplicação, e as conexões entre os componentes de cada tier.

4.2.2.2 View Packet Overview

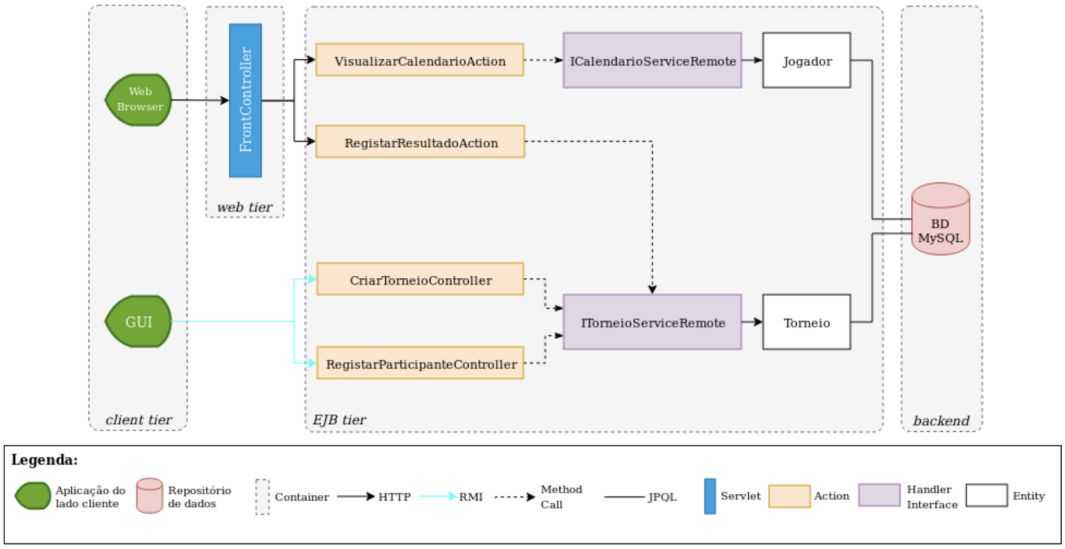

Figure 14: Multi tier

4.2.2.3 Architecture Background

DTO (Data Transfer Object) - objetos utilizados na comunicação entre os tiers Web e Client com o EJB tier, de modo a reduzir a quantidade de dados transmitidos entre estes, já que permitem transferir apenas a informação essencial para realizar uma operação. Isto beneficia o desempenho do sistema.

JPQL (JPA Query Language) - utilizado para procurar objetos na BD, sendo que as queries são escritas ignorando a sintaxe do sistema de gestão da BD. Isto beneficia a portabilidade, como também o potencial de reutilização do sistema, no caso de se querer fazer alterações ao sistema de gestão da BD.

FrontController - padrão utilizado para processamento dos pedidos, e os casos de uso organizados em Servlets (subclasses de Action). Este padrão permite a criação de subclasses para novos casos de uso facilmente, facilitando a extensibilidade.

4.3 Allocation Views

As vistas de alocação focam-se no mapeamento da arquitetura de software para o hardware, de modo a que seja observado e analisado o seu desempenho.

4.3.1 Deployment View

4.3.1.1 View Description

Esta vista escreve o mapeamento dos componentes e conectores para o hardware no qual o software é executado.

4.3.1.2 View Packet Overview

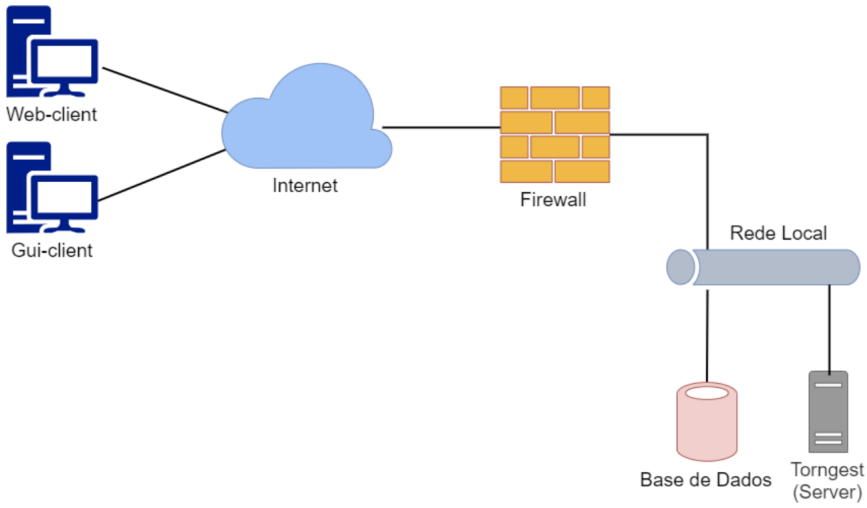

Figure 15: Deployment

4.3.2 Installation View

4.3.2.1 View Description

Nesta vista é descrita o mapeamento de módulos para as pessoas, grupos ou equipas encarregadas do desenvolvimento dos módulos.

4.3.2.2 View Packet Overview

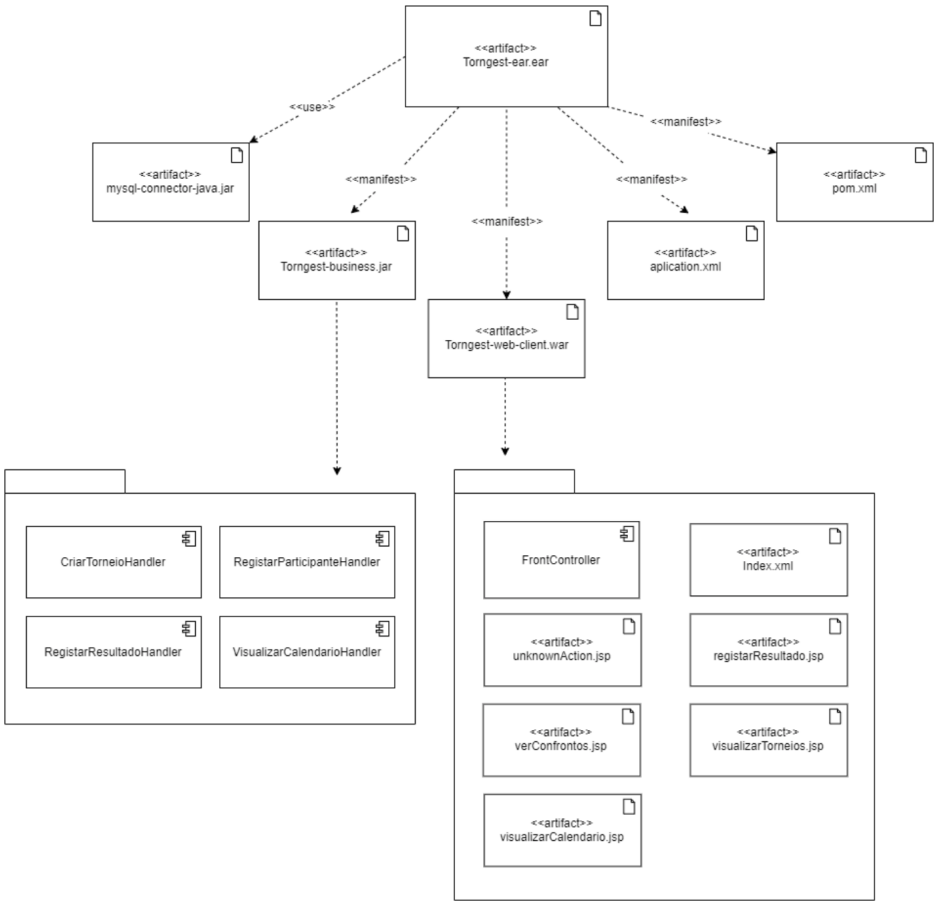

Figure 16: Installation

5 Sonarqube

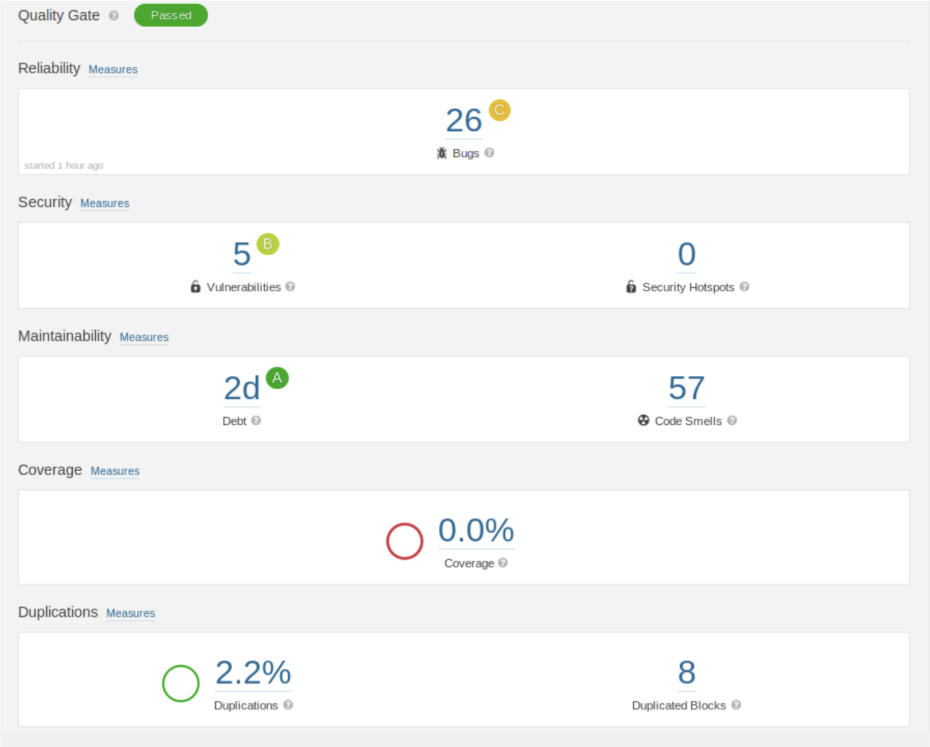

Figure 17: Sonar Qube resultados

Através da ferramenta Sonarqube, com uso de Quality Gate, é possível concluir, primeiro, que o sistema está pronto para produção. No entanto, são observados 26 bugs, 5 vulnerabilidades (código que vulnerável a exploração por atacantes), 57 code smells (código confuso e/ou ambíguo) e 8 blocos duplicados. É estimado, através da observação da Technical Debt, que a correção destes code smells demore 2 dias, nesta fase de produção do sistema.

Em termos de segurança, esta é afetada pelas vulnerabilidades identificadas. A capacidade de manutenção do sistema é afetada pelos code smells. Por fim, os bugs identificados têm uma má influência na fiabilidade do sistema, sendo que esta é a mais profundamente afetada de todos estes requisitos de qualidade, com classificação C.

Bugs:

Os bugs encontrados localizam-se, maioritariamente, no pacote web-client, seguido pelo pacote business. Seguidamente, iremos identificar especificamente os problemas encontrados, divididos por tipo.

Dos 26 bugs, 22 são intermédios (major):

- *Intermediate stream methods* não são utilizados por *terminal stream operations*, deste modo, são obsoletos, porque não são executados.
- *Servlets* não têm campos de instância mutáveis. Deviam ter campos *static* e/ou *final* , de modo a evitar comportamentos inesperados dos servlets em *run-time* porque todas as *threads* partilham instâncias de *servelets* e dos seus campos.
- Falta de identificação de linguagem nos elementos html. Isto é um elemento essencial aos screen-readers.
- Falta de atributos id e scope para elementos ¡th¿, o que afeta a compreensão por parte de screen readers

  E dos mínimos (minor), temos 4:

- Falta de descrição dos elementos de html ¡table¿. Este é outro elemento benéfico para o funcionamento de screen readers;

Code Smells:

Os bugs encontrados localizam-se, maioritariamente, no pacote web-client, seguido pelo pacote business, e por fim, pelo pacote gui-client. Estes também são divididos por tipo:

Dos bloqueadores (blockers), temos 8 dos 57 totais:

- métodos clone *overriden* Isto não deve ser feito, devendo ser usado um construtor com uma cópia ou uma copy factory deste método, porque as regras para fazer *overriding* deste são confusas;
- Na mesma classe pai e filho utilizam o mesmo nome para variáveis diferentes, o que pode confundir os elementos da equipa.

  Dos críticos (*critical*), temos 4:

- Falta de indentação para organizar o código após else, o que pode levar a confusão;
- Métodos de alta complexidade cognitiva (o fluxo de ações é difícil de seguir).

  Dos*major*, temos 19:

- Blocos de código duplicado;
- Método no qual são apenas autorizados 7 parâmetros utiliza 9 parâmetros;
- Variáveis não utilizadas;
- *If Statements* que não são *merged*;

- *Override* de variáveis criadas num *scope* mais geral que aquele em que são utilizadas, o que afeta a capacidade de leitura;
- Classes de *utility* não devem ser instanciadas porque são coleções de membros estáticos. Deste modo, deve ser criado um private constructor que substitua o *public constructor* dentro delas;
- Expressões que são sempre *true*;
- Expressões gerais de exceção são utilizadas em vez de blocos *try/catch* para a exceção específica;
- Anotação @*Override* ão utilizada em algumas operações de *overriding*;
- Atributos deprecados utilizados.

  Por fim, dos minor temos 25:

- Certos nomes de campos não são chamados convencionalmente de acordo com a linguagem (*Java*), com nomes que coincidam com uma expressão regular fornecida;
- Classes que fazem override de clone devem implementar Cloneable e chamar o super.clone();
- Lambdas utilizados, em vez de referências a métodos/construtores;
- URIs (*Uniform Resource Identifier*) que são*hardcoded*, e não obtidos através de parâmetros customizáveis;
- Em vez de utilização de método específico para uma ação (neste caso, utilização de método isEmpty() para verificar se o elemento é vazio) é utilizado código que não é específico para esta verificação (neste caso, size());

5.1 Bug resolution Results

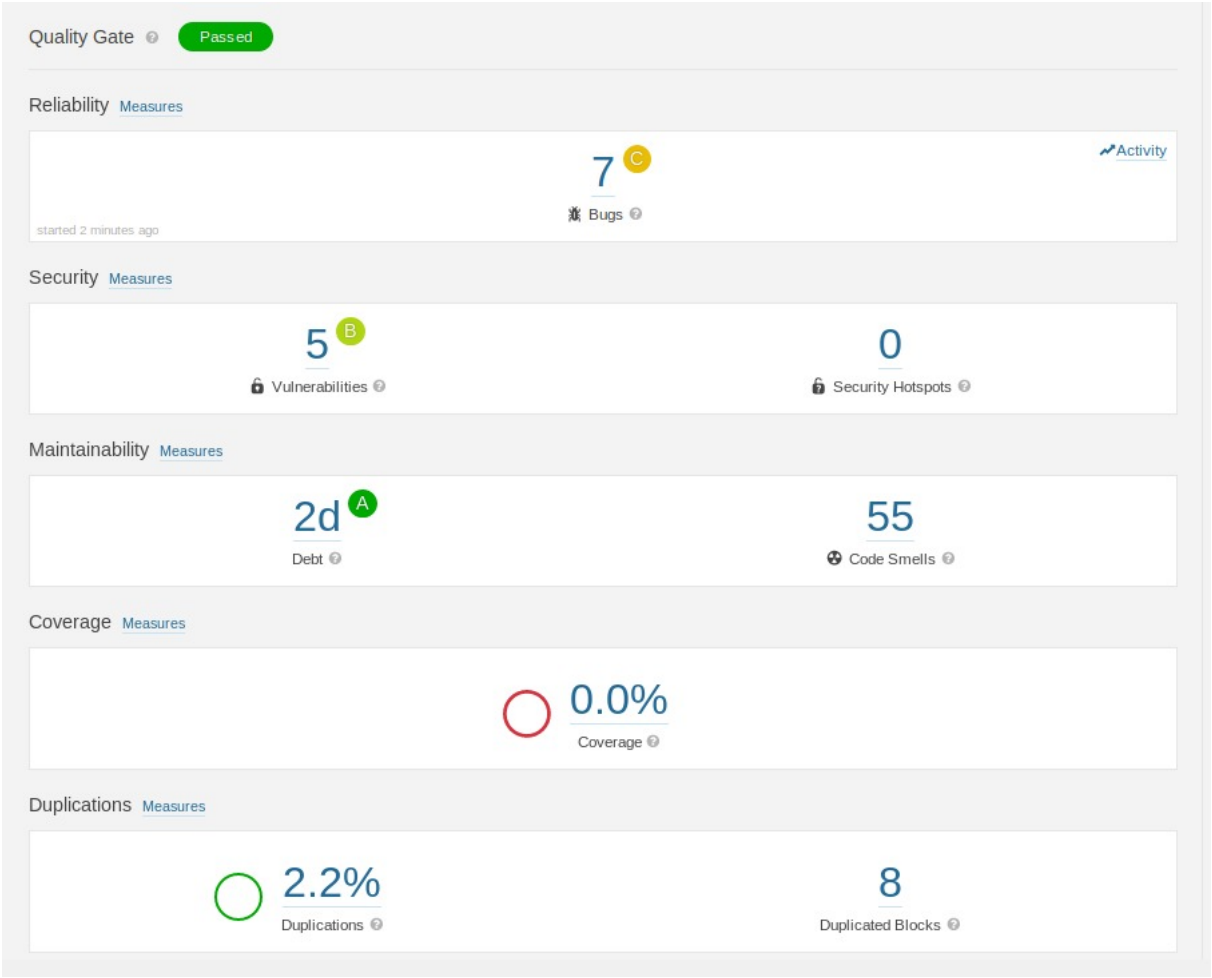

Figure 18: Sonar Qube resultados com bugs resolvidos

Após terem sido aplicadas alterações a algumas classes e documentos (html, jsp) onde foram identificados bugs do tipo *critical* e*blocker*, conseguimos diminuir o número de bugs de 26 para 7, verificando que a fiabilidade do sistema foi aumentada. Isto pôde ser observado, através de nova análise do Sonarqube.

Vulnerabilidades:

As vulnerabilidades encontradas localizam-se, maioritariamente, no pacote web-client, seguido pelo pacote business. Mais uma vez, identificamos os problemas encontrados.

Todas as vulnerabilidades identificadas são mínimas:

- Throwable.printStackTrace(..) é chamado, em vez de um *logger*. Este método imprime um *throwable* e o seu *stack trace System.err*, o qual pode expor informações sensíveis a quem for exposto ao erro que causa o chamamento deste método;
- Falta de blocos *try/catch* nos métodos em *servlet*.A falha de lidar com exceções deixa o sistema num estado vulnerável a ataques de *Denial of Service* ou a exposição de informação sensível.

References

[1] BASS, Len et al. - *Software Architecture in Practice*. 3rd ed. 2013.

[2] CLEMENTS, Paul et al. - *Documenting Software Architectures: Views and Beyond*. 2nd ed. 2011.

[3] RICHARDSON, Michael - *Guideline: Software Architecture Document [Página Web]. [21-11-2019]*. Disponível na www: `http://www.michael-richardson.com/processes/rup_classic/core.base_rup/guidances/guidelines/software_architecture_document_F4C93435.html`

[4] JOHNSON, S. - *Course Registration System Software Architecture Document [Página Web] , 21-03-1999. [21-11-2019]*. Disponível na www: `https://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.html`