Projeto Temático em Desenvolvimento de Aplicações

Relatório Final

Gestão de Tráfego - Radares

Grupo 5: Austin Foote nº84935

Bruno Ferreira nº95291

Carolina Tavares nº92658

Diogo Oliveira nº95182

Maria Nobre nº92659

1ªSemestre Ano Letivo 2019/2020

Projeto Temático em Desenvolvimento de Aplicações

Relatório Final

Gestão de Tráfego - Radares

Grupo 5: Austin Foote nº84935

Bruno Ferreira nº95291

Carolina Tavares nº92658

Diogo Oliveira nº95182

Maria Nobre nº92659

Orientador: Joaquim Ferreira

1ªSemestre Ano Letivo 2019/2020

Resumo

Para este Projeto Temático em Desenvolvimento de Aplicações foi proposto desenvolver uma aplicação de acordo com alguns temas, que, em grupo, foi escolhido o tema “Gestão de Tráfego por Radares”. Recebendo então certos dados recolhidos por um Radar. Sendo assim, decidiu-se desenvolver uma aplicação que permitisse aos clientes, dependendo do sentido, verificar as estatísticas do tráfego, como por exemplo, velocidade média, se existe ou não muito transito no local entre outras informações.

A aplicação teria 2 tipos de utilizadores: sem e com registo. Os sem registo teriam acesso ao trânsito, tráfego na zona e qualquer informação de velocidade que lhes fosse útil caso se quisessem dirigir até ao local desejado. Relativamente aos com registo ou “entidades” que seriam, por exemplo, o governo, não só teriam acesso aos dados que os sem registo têm, como também, teriam acesso a um histórico de velocidades, para fins estatísticos, sobre os carros que passam naquele local.

Para isto foram estabelecidos alguns requisitos funcionais e não funcionais importantes para o projeto. De seguida foram criados alguns diagramas que ajudassem o planeamento e gestão deste projeto, sendo criados diagrama de classes, diagrama de atividades, diagrama de entidades, diagrama de caso de usos e diagrama de componentes.

Com estes diagramas bem estabelecidos programou-se e implementou-se o que tinha sido anteriormente estabelecido, criando uma base de dados, a partir de um ficheiro de excel com informações recolhidas por um radar, que continha dados sobre velocidades, o tipo de veículos na via, entre outros. Sendo assim, aos poucos foi criada a conexão da base de dados à aplicação servidor e através de sockets e multithreading, aos respetivos clientes.

Neste relatório serão abordados e aprofundados todos estes tópicos e passos, apresentando o planeamento, desenvolvimento e as conclusões deste projeto.

Índice

Índice de figuras

Índice de tabelas

1 Introdução

No âmbito do Projeto Temático em Desenvolvimento de Aplicações, foi proposto ao grupo conceber uma aplicação, cujo tema escolhido foi “Gestão de Tráfego por Radares”. Este tema consiste na utilização dos dados fornecidos por radares para gerar estatísticas relevantes aos condutores e outras organizações interessadas em estudar o tráfego na zona.

Os dados serão fornecidos ao grupo pelo gestor dos radares da Barra e Costa Nova, que, por sua vez, são colocados numa base de dados criada pelo grupo. A aplicação irá utilizar estes dados para gerar e mostrar estatísticas, tais como velocidade média e contagem de veículos. Com estes dados, os utilizadores da app poderão determinar se existe trânsito na zona ou não.

Estas estatísticas também poderão ser partilhadas com mais detalhe para certas organizações, através de uma interface de login, sendo que as credenciais são fornecidas pelo administrador do sistema.

Estando este projeto incluído no módulo Temático em Desenvolvimento de Aplicações, juntamente com as disciplinas de Engenharia de Software e de Sistemas de Bases de Dados, tinha-se também, como objetivo, aplicar os conhecimentos adquiridos nestas aulas.

Ao longo deste relatório será então descrito todo o processo de desenvolvimento deste projeto, desde o seu planeamento, definição dos requisitos, planificação de tarefas, até à implementação da aplicação.

Para finalizar, será feita uma análise dos resultados, uma pequena reflexão crítica e conclusão.

1.1 Visão geral do sistema

Foram colocados vários radares, na Barra e Costa Nova, estes radares detetam os objetos que estão a passar na zona. Identificam os objetos distinguindo se se trata de um carro, de um veículo pesado, de uma bicicleta/mota ou de um peão. Outros objetos ficam na categoria de objetos não identificados, por exemplo, animais, árvores, etc. O radar consegue acompanhar estes objetos durante um certo troço da estrada, fazendo mais que uma medição nesse troço, sendo que quanto mais tempo acompanhar o objeto mais correta será a medição. Para além da sua categoria, de cada objeto são guardadas mais informações, nomeadamente, a data, hora e zona (do radar) da medição, o sentido e a que velocidade vai. Assim, sabendo quantos carros é que estão naquela zona e a que velocidade vão, percebesse se existe congestionamento na zona.

Os radares após captarem estes dados guardam-nos num ficheiro de texto. O grupo vai trabalhar com os dados a partir destes ficheiros, com o objetivo de criar uma aplicação que informe os seus utilizadores destas várias estatísticas. O grupo configurará este sistema utilizando comunicação por sockets, com um modelo cliente-servidor. O sistema tem de ser configurado de modo a saber quantos radares existem, onde é que se encontram e quantos sentidos e vias existem na estrada. Os dados dos ficheiros provenientes do radar são colocados numa base de dados própria, que depois será usada pela aplicação, esta tem de ser configurada de modo a fazer os cálculos necessários para mostrar as várias informações num dado intervalo de tempo.

Relativamente à aplicação esta terá dois tipos de utilizadores, os com e os sem registo, assim, tem de se criar uma lista dos utilizadores com os seus respetivos logins e passwords. As informações que cada um dos tipos de utilizadores terão acesso não serão as mesmas, pois as suas necessidades também serão diferentes.

O utilizador sem registo será, por exemplo, um condutor normal que quer ir passear e procura saber se ir para a praia será uma boa ideia naquela hora ou se estará muito trânsito. Assim, ele entra na aplicação rapidamente (sem precisar de fazer login), procura a informação que necessita, que neste caso, é a contagem dos carros e a velocidade média no momento. Sabendo isto, ele depois toma a decisão de ir ou não para a praia.

O utilizador com registo, por exemplo o estado, pretende perceber se aquela zona necessita de melhorias no que toca ao controlo do congestionamento. Eles irão entrar na aplicação, usando credenciais fornecidas pelo administrador, após terem-se registado previamente. De seguida, eles irão consultar o histórico das velocidades. Para perceberem quais as horas e dias em que houve mais movimento e qual a quantidade de carros que circulam na zona. Assim sabem se será

necessário tomar certas medidas para melhorar a circulação nestas vias, melhorar a forma de entrada na zona, ou então irão usar estes dados apenas para efeitos estatísticos.

1.2 Cliente

Para este projeto, o cliente é a empresa que contratou o grupo para criar esta aplicação. Esta empresa pretendia que fosse criada uma aplicação que fizesse a gestão do tráfego nas diversas zonas onde estão colocados os seus radares. Deixando a critério do grupo quais as estatísticas a serem geradas, quais os seus requisitos e casos de utilização. O cliente não necessita de ter acesso a como são geradas as estatísticas, apenas precisa de configurar a aplicação para funcionar para cada um dos diferentes radares.

1.3 Objetivos

Assim, tendo em conta a visão geral deste projeto podem-se agrupar os seguintes objetivos:

- Conceber uma aplicação que permita a gestão do tráfego para qualquer zona à escolha do administrador;
- Permitir a qualquer condutor ter a possibilidade de tomar decisões com base na informação do trânsito obtida pela aplicação;
- Permitir a certas entidades o acesso a informações mais detalhadas do trânsito na zona, para estudos estatísticos ou, eventualmente, para melhorar a circulação do trânsito na área.

2 Planeamento

Relativamente ao planeamento do projeto, optou-se por seguir um método ágil. Sendo assim, o planeamento foi dividido em várias iterações. Para cada uma das iterações, iam sendo definidas várias tarefas e as suas respetivas durações estimadas. Ou seja, logo no início do projeto, definiram-se as primeiras tarefas da primeira iteração, definindo-se também uma data limite para o cumprimento daquela iteração. Foram feitas iterações de aproximadamente 1 ou 2 semanas, tirando a última que foi de 4 semanas.

O grupo reuniu-se com o orientador quase todas as semanas, nessas reuniões, não só se discutia o que já tinha sido feito anteriormente como também se discutia quais deviam ser os passos seguintes. Ajudando assim a perceber se a iteração estava a ser bem planeada e no que consistiria a próxima. Assim, foi obtido um único cronograma que ia sendo ajustado ao longo do projeto, ou seja, iam sendo acrescentadas as várias iterações planeadas. Este cronograma pode ser consultado no Anexo A – Planeamento, do relatório e também no ficheiro Excel entregue em anexo, este ficheiro possui também uma tabela com todas as tarefas definidas.

Nas primeiras fases, durante o planeamento, definem-se os objetivos do trabalho, faz-se o levantamento dos casos de utilização e dos requisitos da aplicação. Elaboram-se vários diagramas. Começando-se pelo diagrama de casos de utilização, o diagrama de classes, com os seus respetivos atributos e métodos mais relevantes e o diagrama de entidades, o diagrama físico da base de dados. Usando um método ágil, a implementação do projeto deve ser dirigida por testes, assim feito o planeamento passa-se à criação dos casos de teste. Ao criarem-se os casos de teste, definem-se os métodos, isto é, define-se o que cada método deve fazer, quais devem ser os resultados esperados tendo em conta os parâmetros introduzidos. De seguida, com base no que se definiu, procede-se à implementação e programação das várias classes, métodos, interfaces e criação da base de dados. Tendo isto em consideração, o grupo tentou seguir todos estes passos de um método ágil, passando por todas as suas fases.

Decidiu-se ainda que a partilha da documentação iria ser feita usando as ferramentas “Git” e “GitHub”. O “Git” é um sistema de controle de versões, permite desenvolver projetos com várias pessoas em simultâneo, editando e criando arquivos, sem o risco de suas alterações serem sobrescritas. O “GitHub” usa o “Git” e permite a criação de repositórios públicos ou privados. Com o “GitHub” é possível verificar com facilidade quais as mudanças que foram efetuadas no projeto e ainda quem as fez. Assim, foi criado um repositório para o grupo, todos os membros tinham acesso a este repositório e era nele que se colocava toda a documentação relativa ao projeto.

3 Modelo de requisitos

3.1 Requisitos funcionais

Para qualquer projeto é crucial definirem-se as funcionalidades do sistema a desenvolver. Assim, estabelecida a visão geral do sistema e quais os seus objetivos, passou-se para a definição dos requisitos, começando então pelos requisitos funcionais.

A seguinte tabela, tabela 1, trata-se de um inventário das funções que o sistema tem de suportar e das suas prioridades.

Tabela 1 - Requisitos Funcionais

Refª	**Requisito funcional**	**Prioridade**
RF.1	Permite a consulta das informações relativas às velocidades das viaturas em ambos os sentidos (histórico de velocidades).	+++
RF.2	Permite a consulta da contagem das viaturas que passam pelo radar (tráfego).	+++
RF.3	Permite a consulta da velocidade média em intervalos de 10 em 10 minutos.	+++
RF.4	Sistema indica a velocidade máxima e mínima.	++
RF.5	Os condutores (sem registo) só terão acesso à velocidade média de trânsito (de 10 em 10 min) e ao contador de viaturas.	+++
RF.6	As entidades registadas (Governo, IMT, Brisa…), terão acesso a todas as estatísticas disponíveis.	+++
RF.7	Permitir a configuração da aplicação, de modo a funcionar para qualquer radar, independentemente da sua localização.	-
RF.8	Permitir o registo e eliminação de entidades.	++

3.2 Restrições e requisitos não funcionais

3.2.1 Requisitos de interface e facilidade de uso

Definidos os requisitos funcionais, passaram-se para não funcionais. Assim, na tabela abaixo, tabela 2, estão apresentados os requisitos que não são essenciais para o funcionamento do serviço, mas que facilitam a navegação da interface para o utilizador.

Tabela 2 - Requisitos de interface e usabilidade

Ref[a]	**Requisito de interface e usabilidade**
RInt.1	A interface do condutor é a interface principal, sendo logo mostrada quando se abre a aplicação.
RInt.2	A interface do condutor possui um mapa com a localização do radar, tendo também indicação do número de vias e sentidos nessa zona.
RInt.3	Mapa tem de ser manipulável, mas será possível afastar-se muito da zona do radar.
RInt.4	A interface do condutor possui também uma tabela com as informações da velocidade média para cada um dos sentidos, a informação da contagem dos carros, mostrando apenas a diferença e a avaliação do trânsito (muito trânsito, trânsito moderado e pouco trânsito).
RInt.5	A interface das entidades possui uma tabela com as velocidades máxima e mínima e uma tabela com um histórico de todas as velocidades.
RInt.6	O administrador tem ainda acesso a uma interface de configuração onde faz o registo das novas entidades e as outras configurações da aplicação.
RInt.1	A interface do condutor é a interface principal, sendo logo mostrada quando se abre a aplicação.

4 Modelo de Casos de Utilização

4.1 Visão geral

O primeiro diagrama que se deve fazer é o diagrama de casos de utilização. Para criar este diagrama é necessário saber os cenários de utilização da aplicação. Estes cenários já foram definidos previamente e foi a partir deles que se criou o seguinte diagrama apresentado na figura 1.

Como já foi referido, esta aplicação iria a partir dos dados recebidos pelo radar gerar estatísticas que iram depois ser consultadas pelos condutores, sendo que estes podem estar ou não registados. A aplicação teria um administrador encarregue da sua configuração. Assim, os casos de uso mais importantes são a criação das estatísticas, a consulta das mesmas, a autenticação e a configuração do sistema.

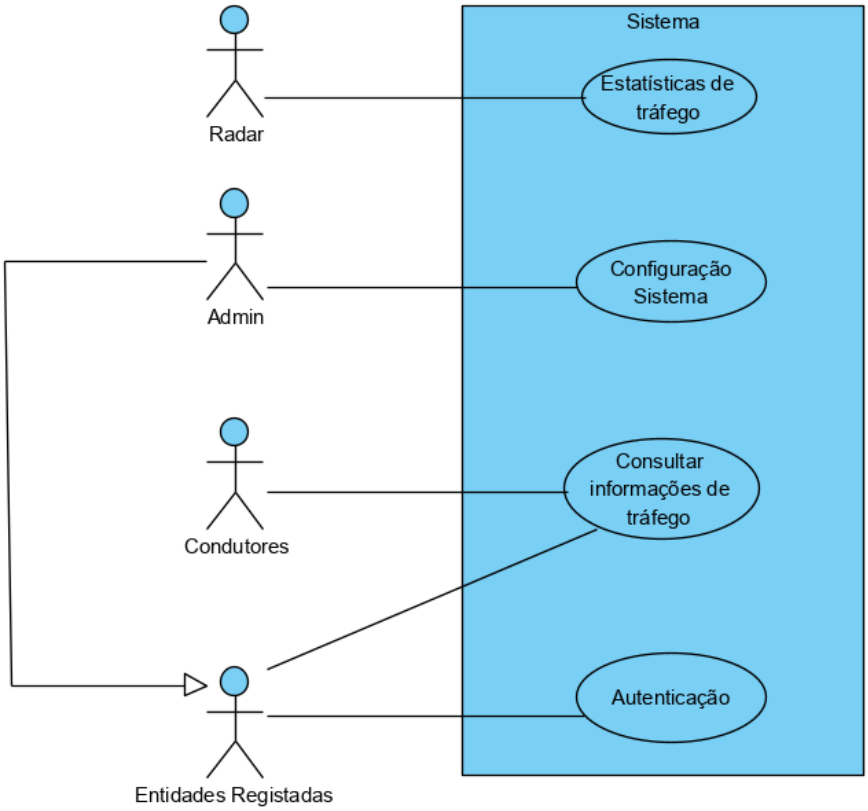

Figura 1 - Diagrama de Casos de Utilização

4.2 Atores

A construção de um diagrama de casos de utilização envolve também a definição dos atores que estão envolvidos com o sistema. Cada um deles interage com a aplicação/sistema através de pelo menos um dos casos de utilização representados. Estes atores estão descritos na tabela 3.

Tabela 3 - Descrição dos atores

Ator	Descrição
Radar	Equipamento que recolhe dados brutos do tráfego na zona.
Administrador	Indivíduo que configura o sistema e faz a sua manutenção, especialização das Entidades Registadas, herdando as suas características.
Condutores	Utilizadores normais que pretendem fazer consultas simples de tráfego.
Entidades Registadas	Utilizadores com login que têm acesso a informações mais detalhadas.

4.3 Descrição dos casos de utilização

De seguida, para uma melhor e mais detalhada explicação do diagrama de casos de utilização e do que este representa, passar-se-á para uma descrição de cada caso de utilização. Sendo que, para os casos em que seja pertinente, isto é, para os casos mais complexos, serão apresentados diagramas de atividade. Os diagramas de atividades são fluxogramas que decompões uma tarefa/atividade em subactividades, ajudando assim a explicar no que consiste o caso e quais as interações dos atores com o sistema.

4.3.1 [Estatísticas de Tráfego #1]

Tabela 4 - Caso de Utilização #1 – Estatísticas de Tráfego

Nome:	Erro! Utilize o separador Base para aplicar Heading 3 ao texto que pretende que apareça aqui.
Atores:	Radar (inicia)
Prioridade:	Alta
Finalidade:	Processar os dados do radar para gerar estatísticas de tráfego.
Pré-condições:	Receber dados provenientes do Radar.
Sumário:	Os dados fornecidos pelo radar serão processados pelo sistema de forma a criar estatísticas úteis. Vão ser utilizadas as velocidades para saber o sentido das viaturas (velocidade negativa e positiva referem-se a sentidos diferentes) e para calcular as velocidades médias de 10 em 10 minutos. É, também, criado um histórico de velocidades e vai ser guardada a velocidade máxima e mínima desde sempre. Vão ser utilizados os IDs dos objetos para fazer a contagem das viaturas de cada um dos sentidos, calculando também a sua diferença, dando a indicação de quantas viaturas estão naquela zona.

4.3.1.1 Diagrama de Atividades

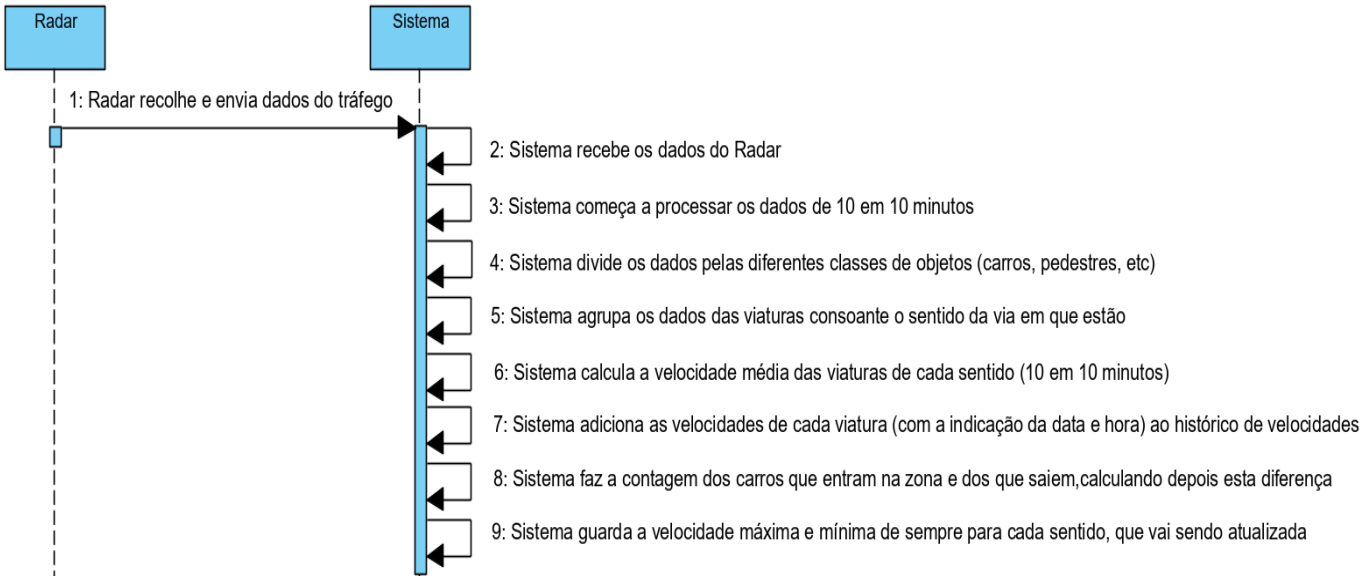

Figura 2 - Diagrama de Atividades – Estatísticas de Tráfego

4.3.2 [Configuração Sistema #2]

Tabela 5 - Caso de Utilização #2 - Configuração Sistema

Atores:	Administrador(inicia)
Prioridade:	Baixa
Finalidade:	Configuração da aplicação para cada radar, escolhendo que dados vão ser exibidos.
Pré-condições:	Possuir aplicação e interfaces criadas e uma interface de configuração.
Sumário:	O administrador configura o sistema para cada radar, podendo usar esta aplicação independentemente de qual o radar e de qual a sua localização. Administrador tem de indicar quantos radares existem e onde estão, quantas vias existem para cada um dos sentidos. Feita a definição da localização do radar, configura quais as estatísticas que vão estar disponíveis para cada um dos tipos de utilizadores da aplicação. Terá também uma interface para fazer o registo das entidades.

4.3.2.1 Diagrama de Atividades

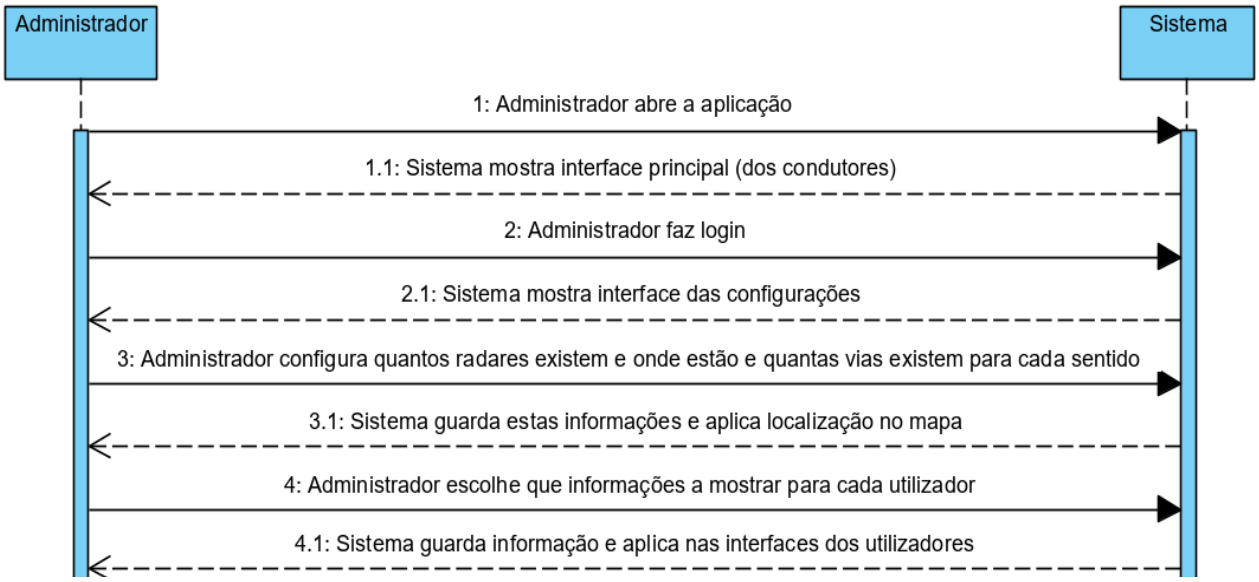

Figura 3 - Diagrama de Atividades – Configuração Sistema

4.3.3 [Consultar informações do tráfego #3]

Tabela 6 - Caso de Utilização #3 - Consultar Informações do Tráfego

Atores:	Condutores e Entidades registadas (inicia)
Prioridade:	Alta
Finalidade:	Consultar a interface principal da aplicação.
Pré-condições:	Interfaces criadas, sistema a gerar estatísticas e configuração do sistema feita.
Sumário:	Utilizador abre a aplicação. Sistema mostra a interface principal, que corresponde à interface com as informações para os condutores, utilizadores sem registo, ou seja, são as estatísticas menos detalhadas. No caso dos condutores não farão mais nada, no caso das entidades registadas podem prosseguir à autenticação.

4.3.4 [Autenticação #4]

Tabela 7 - Caso de Utilização #4 - Autenticação

Atores:	Entidades Registadas (inicia)
Prioridade:	Alta
Finalidade:	Permitir o acesso a outras informações que utilizadores sem login não podem ver.
Pré-condições:	Entidades têm de fazer um registo prévio contactando o administrador diretamente.
Sumário:	Utilizador abre a aplicação. Sistema mostra a interface principal. Utilizador escolhe a opção de fazer login. Sistema mostra a interface de autenticação. Utilizador insere as credenciais. No caso do administrador, ele é redirecionado para a sua interface própria de configuração, se for uma

	entidade registada, mostra a interface destes utilizadores, que possui mais informações estatísticas que a do condutor.

4.4 Cobertura de requisitos

Cada caso de utilização implementa ou pelo menos está relacionado com algum dos requisitos funcionais. Assim, na tabela 8 está representada essa mesma relação dos casos de utilização e os requisitos.

Tabela 8 – Cobertura dos requisitos

	RF1	**RF2**	**RF3**	**RF4**	**RF5**	**RF6**	**RF7**	**RF8**
Estatísticas de Tráfego #1	X	X	X	X				
Configuração Sistema #2					X	X	X	X
Consultar informações do tráfego #3	X	X	X	X	X			
Autenticação #4						X	X	X

5 Diagrama de Classes

A partir do diagrama de casos de utilização, definiram-se os principais conceitos deste sistema e elaborou-se o diagrama de classes. Para cada classe definiram-se os atributos e métodos que fossem necessários a implementar. A criação deste diagrama demorou algum tempo e até à fase da implementação, este foi ajustado e corrigido várias vezes até ser obtido o diagrama apresentado na figura 4.

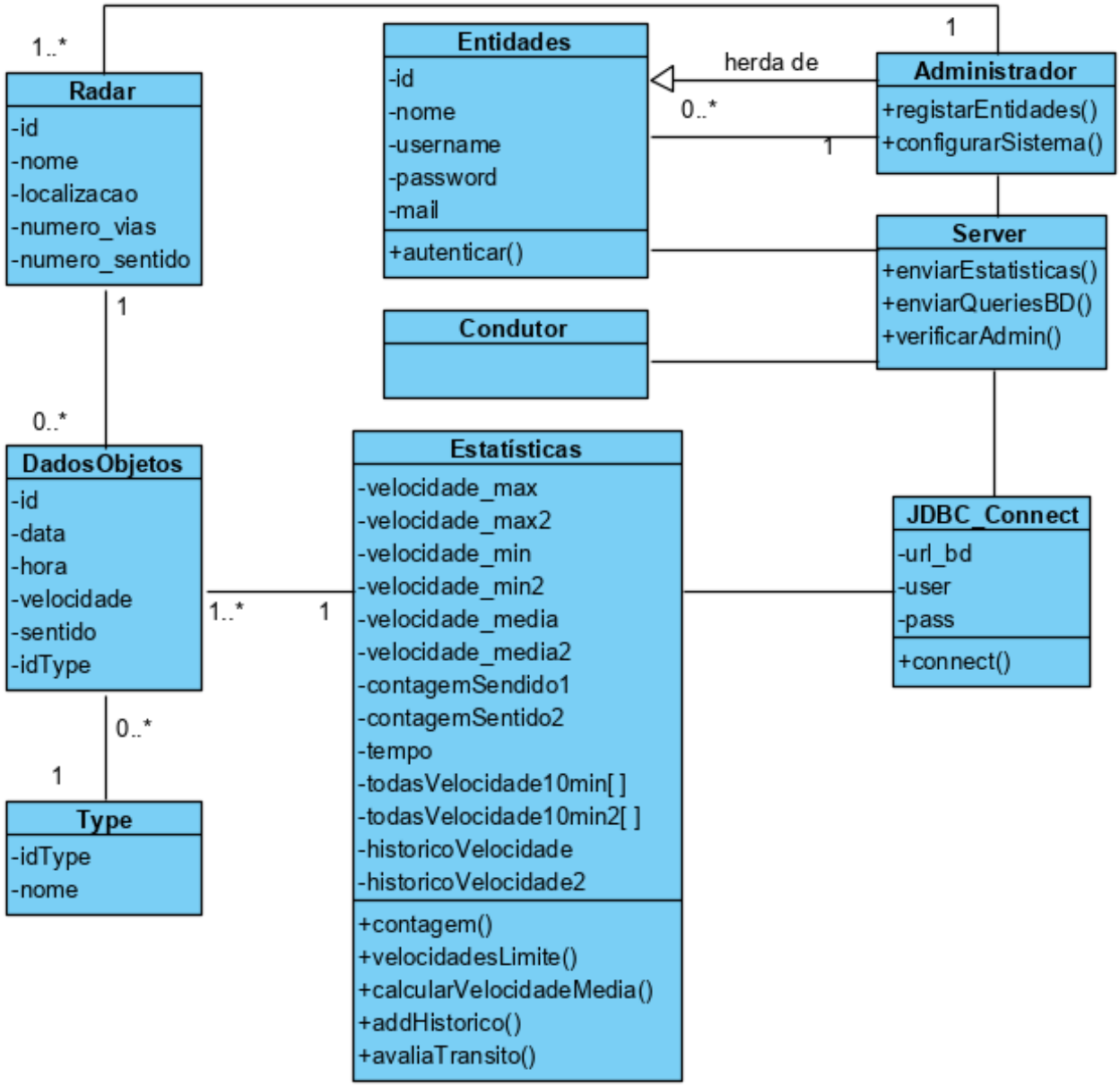

Figura 4 - Diagrama de Classes

De seguida, será feita uma pequena descrição de cada uma das classes presentes no diagrama de classes:

- **Radar -** Esta classe representa cada radar, sendo que cada um está associado a um ID. Aqui pode-se encontrar também como atributos, o seu nome e localização, o número de vias e sentidos que ele monitoriza. Cada Radar tem um Administrador e cria vários objetos da classe DadosObjetos.
- **DadosObjetos -** Esta classe representa os objetos lidos pelos radares. Ou seja, cada objeto desta classe é um objeto que passou pela estrada, tendo um ID, um registo da hora, velocidade, sentido e tipo (carro, mota, etc.). Cada objeto desta classe tem um só Radar, um só Type.
- **Type -** Como os radares conseguem distinguir os diferentes tipos de objetos que passam por eles, foi criada esta classe para guardar o nome da classificação dos objetos, associando esse nome a um ID próprio. Cada uma destas classificações pode estar associada a vários objetos da classe DadosObjetos.
- **Estatísticas** - Esta classe representa as estatísticas geradas com base nos vários objetos da classe DadosObjetos. Assim esta classe tem como atributos o tempo, e para cada sentido, um array com todas as velocidades de 10 minutos, a contagem, a velocidade média, máxima e mínima e um histórico com todas as velocidades. Esta classe tem ainda vários métodos associados, que servirão para gerar as estatísticas.
- **Entidades –** Esta classe representa os utilizadores com registo. Tendo como atributos, o id, o nome, *username*, email e password. Sendo esta classe um cliente, está associada à classe Server.
- **Administrador** – É uma especialização da classe Entidades, herdando todos os seus atributos e métodos. Contudo, terá algumas particularidades próprias, como é o caso dos métodos a mais que possui.
- **Condutor -** É uma classe que representa os utilizadores não registados, sendo também um cliente estará associada à classe Server. Esta classe não tem atributos pois deste utilizador não será guardada nenhuma informação.
- **Server -** Esta classe que representa o servidor da aplicação, estando associada às classes que representam os utilizadores da aplicação.
- **JDBC_Connect -** Esta classe servirá para estabelecer a conexão entre o servidor e a base de dados, tendo como atributos o url_bd, o user e a pass da base de dados.

6 Diagrama de Entidades

Para a criação do diagrama de entidades, primeiro analisou-se o diagrama de classes de modo a perceber-se quais as classes que iam necessitar de estar presentes na base de dados. Teve-se ainda de perceber se eram precisas mais classes, de modo a garantir que a base de dados cumpria as regras e que esta era criada corretamente.

O diagrama de entidades também foi sendo ajustado, mas conclui-se que o diagrama da base de dados seria o que está apresentado na figura 5.

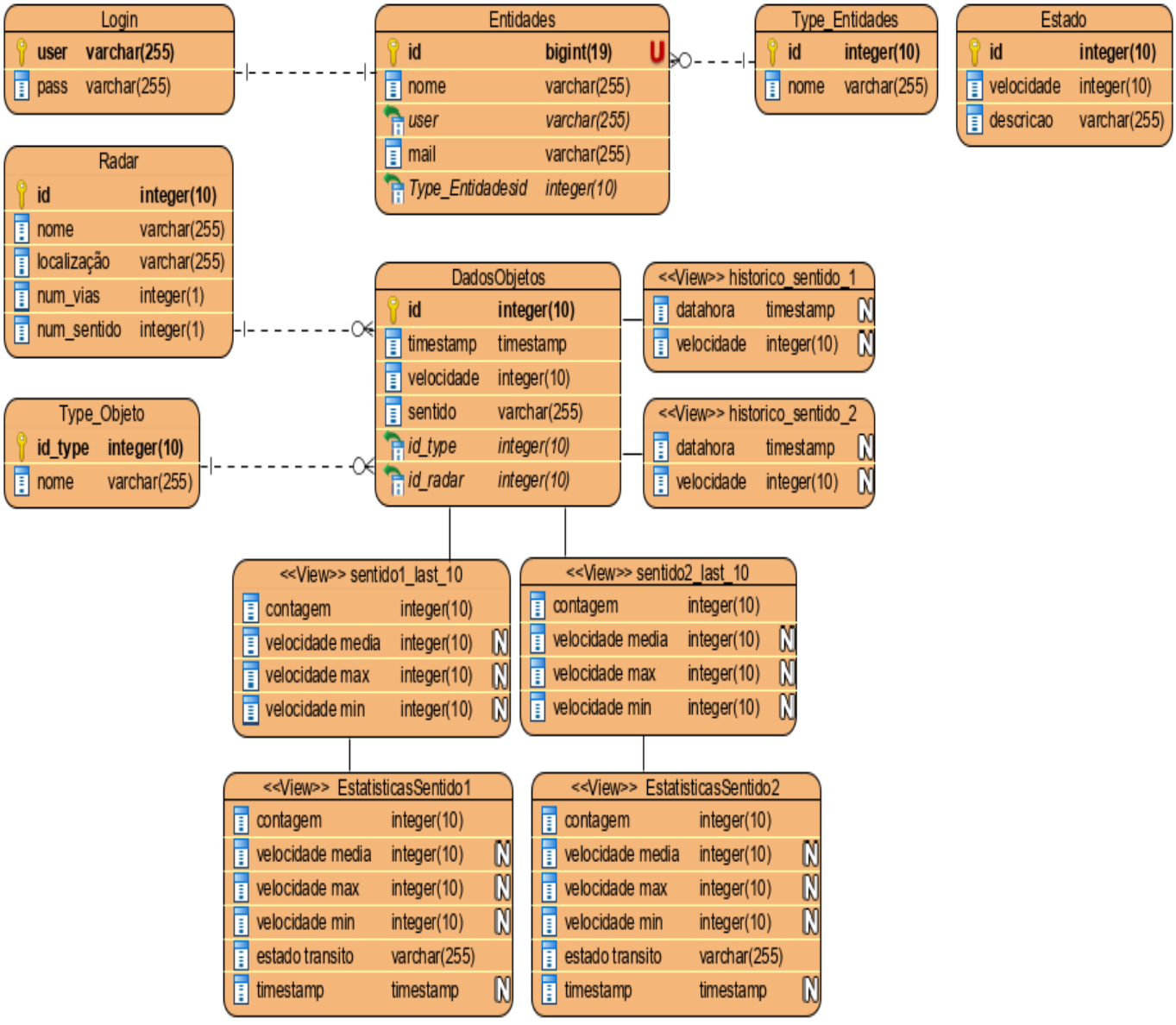

Figura 5 - Diagrama de Entidades

Como se pode verificar, existem várias tabelas que são comuns ao diagrama de classe, mas existem algumas que foram acrescentadas apenas aqui. De seguida, tentar-se-á esclarecer a razão destas decisões.

As tabelas que são comuns ao diagrama de classes são:

• **Radar** – Esta tabela guarda as informações dos vários radares. Cada Radar possui como chave primária um ID único. Tem ainda, o seu nome e localização como varchar, e o número de vias e sentidos que monitoriza como integer.

• **DadosObjetos** – Esta tabela guarda todos os dados registados pelos radares. Cada objeto tem como chave primária um ID único. Existindo ainda o registo da hora, como timestamp, a velocidade, como integer, e o sentido como varchar. Tem ainda como chaves estrangeiras, as chaves primárias das tabelas Radar, id_radar, e Type_Objeto, id_type. Uma vez que, cada objeto é lido por um certo radar e pertence a um dos tipos de objetos.

• **Type_Objeto** – Esta tabela guarda o nome das classificações dos objetos e associa-os a um ID, id_type, que serve com chave primária da tabela.

• **Entidades** – Nesta tabela estão registadas as informações das entidades registadas. Estas informações incluem o nome, username, email, como varchar, e tipo de entidade (Type_Entidadeid) e um ID único como integer.

Não foi criada uma tabela para o Administrador, invés disso, acrescentou-se uma tabela:

• **Type_Entidades** – Aqui pode se encontrar o significado do tipo de entidade registado na tabela de entidades, sendo 0 utilizador administrativo e 1 um utilizador não administrativo. Desta forma a tabela Entidades recebe a chave primária desta tabela, que é o id, como chave estrangeira. Assim, mesmo sem a tabela Administrador, já é possível distinguir facilmente quem são os administradores dentro da tabela das Entidades, o que é útil pois estes terão acesso a mais ferramentas na interface a desenvolver.

Por motivos de segurança, foi também criada uma tabela Login:

• **Login** – Esta tabela guarda o username e password das entidades, sendo o user a chave primária. Desta forma, é possível manter as passwords numa tabela separada da tabela das Entidades, mas existe na mesma uma relação, sendo que as entidades recebem como chave estrangeira o username.

Criou-se ainda uma outra tabela, que iria servir para guardar as definições dos estados de transito, para depois ser feita a avaliação do trânsito:

• **Estado** – Esta tabela associa um certo limite de velocidade a uma descrição da avaliação de trânsito e depois cada classificação é identificada por um id. Por exemplo, se a velocidade for inferior a 20 km/h está muito trânsito.

No diagrama de classes, definiu-se também uma classe chamada Estatísticas, nesta classe, iriam estar todas as estatísticas geradas a partir dos dados recolhidos. Contudo, para a base de dados optou-se por invés de numa tabela, criar estas estatísticas usando vistas materializadas. Uma vista materializada representa o resultado de uma consulta, que será atualizado periodicamente a partir das tabelas originais.

Assim, para gerar as estatísticas necessárias iriam ser criadas vistas materializadas a partir dos dados da tabela DadosObjetos. Foram divididas as estatísticas de cada sentido em vistas diferentes para mais fácil distinção. A seguir serão descritos os motivos de criação das várias vistas. Para simplicidade, só se vai falar de um dos sentidos:

- **histórico_sentido –** Esta vista serve de histórico para velocidades registados pelo radar, estando cada registo associado a uma *timestamp* (Data e hora) e a velocidade.
- **sentido_last_10 –** Nesta vista é guardada a contagem e a velocidade média, máxima e mínima das viaturas que passaram pelo radar nos últimos 10 minutos. Esta tabela atualiza-se a cada 10 minutos.
- **estatisiticas_sentido -** Esta vista serve para fazer a avaliação do estado do trânsito, utilizando as velocidades médias e contagens registadas na vista **sentido_last_10**. Assim, são guardados os mesmos campos da vista sentido_last_10 mas também o estado de trânsito correspondente. Esta tabela também se atualiza a cada 10 minutos.

7 Implementação da Base de Dados

Após a criação do Diagrama de Entidades (figura 5), prosseguiu-se à implementação da Base de Dados. Para a criação da base de dados utilizou-se o sistema PostgreSQL e a plataforma pgAdmin, dois softwares open source potentes que utilizam principalmente programação SQL para a realização das suas tarefas. Foram escolhidos estes softwares pela sua popularidade e pelas suas facilidades de uso, já que o pgAdmin permite fácil manipulação das tabelas e dos seus dados. O código para a implementação da base de dados poderá ser encontrado num ficheiro em anexo "ScriptSQLCriaçãoDB"

7.1 Criação das tabelas

No diagrama de entidades definiram-se as seguintes tabelas: **Radar, DadosObjetos, Type_Objeto, Entidades, Type_Entidades, Login** e **Estado.** A criação da base de dados foi feita, naturalmente, a partir do que tinha sido planeado. Assim, iriam ser implementadas todas estas tabelas e as relações entre elas, colocando as devidas chaves primárias e estrangeiras.

Contudo, a base de dados acabou por sofrer algumas alterações, nomeadamente pelo facto de não se ter criado a tabela Estado. Esta tabela iria servir para guardar as definições da avaliação de trânsito, porém esta funcionalidade acabou por apenas ser implementada nas vistas materializadas. Assim, como a tabela não estava a ser usada, foi retirada da base de dados. Fez-se ainda uma alteração na tabela DadosObjetos, passando o campo sentido de varchar para integer.

7.2 Criação das views

Todas as vistas materializadas que tinham sido definidas foram criadas na base de dados. Contudo, como o grupo não teve acesso a uma *live datastream* dos dados de um radar, não estariam sempre a ser recebidos dados na base de dados. Desta forma, as vistas que geram as estatísticas dos últimos 10 minutos, nunca gerariam nada. Teve então de se criar vistas que gerassem estatísticas, para que se pudesse testar o código na aplicação, utilizando unicamente ao ficheiro de excel fornecido. Assim, foram criadas as seguintes vistas para cada sentido:

- **sentido_all –** Esta vista equivale à vista sentido_last_10, mas invés de ser apenas com os dados dos últimos 10 minutos, são usados os dados do último ano.
- **sentido –** Esta vista equivale à vista estatisiticas_sentido, mas invés de usar a vista sentido_last_10, é criada a partir da vista sentido_all.

7.3 Filtração e inserção dos dados do radar na base de dados

Para se poder testar a aplicação, foi fornecido um ficheiro Excel pelo orientador do grupo, que possuía os registos obtidos por um radar na Costa Nova num intervalo de 4 horas.

Este ficheiro foi analisado pelo grupo e alterado de forma a ser facilmente importado para a tabela **dadosobjetos**. Foram retiradas do ficheiro as colunas de '*UMRRID*', '*BusNo*', '*MilliSecs*', '*Seconds'*, '*Speed [m/s]*' e '*L'* pois foram considerados informações desnecessários, já que não iriam ser utilizados nas tabelas nem mostrados na interface da aplicação. Após isto, foram retirados os registos cujos valores não fossem '0' das colunas '*ML/(Sensor)Zone'* e '*MesaZone'* para que não houvesse registos duplicados de objetos, já que o mesmo objeto poderia ser detetado em duas zonas diferentes ao mesmo tempo.

Como os '*ObjectID's* repetiam a cada 255 registos, foram renumerados de forma a que isto não acontecesse. Foi ainda adicionada uma coluna que distinguisse o sentido do objeto, baseado no sinal das velocidades associados, sendo '0' associado a velocidades negativos, e '1' associado a velocidades positivos. Por último, adicionou-se uma coluna 'id_radar', existindo nesta coluna unicamente o valor '1', já que todos os registos eram do mesmo radar.

Após a filtração dos dados, o ficheiro foi exportado para formato CSV e importado na tabela **dadosobjetos** através do pgAdmin.

8 Métodos e Testes unitários

8.1 Testes Unitários

Testes de Unidade ou Testes Unitários refere-se à fase de testes onde cada unidade do sistema é testada individualmente, ou seja consiste na verificação da menor unidade do projeto de software. *Unit*, é o nome genérico para qualquer estrutura de testes automáticos unitários.

O objetivo é isolar cada parte do sistema para garantir que estão a funcionar conforme foi especificado. Através da utilização deste tipo de testes ao longo da implementação é possível reduzir a quantidade de bugs na aplicação final. Os testes unitários funcionam através de comparação de resultados das funções a serem testadas com valores esperados. JUnit é uma ferramenta que facilita o desenvolvimento e execução de testes unitários em Java. Sendo então muito útil para a criação destes testes em qualquer aplicação Java.

Neste caso não seria apenas necessário criar testes só com o JUnit, pois, como iria existir uma conexão com a base de dados, os testes tinham de ser feitos de modo a não estarem sempre a conectar com a base de dados. Para isso foi encontrada uma solução: “JMock”.

JMock é uma biblioteca que auxilia o Desenvolvimento Dirigido por Testes (DDT) através de objetos mock. Com esta ferramenta podem ser criadas umas espécies de bases de dados falsas específicas para os testes que se querem testar.

Contudo, o grupo não conseguiu utilizar o JMock, decidindo então procurar outras soluções tais como “HSQLDB”, “DBUnit”, “NBUnit” mas, mesmo assim, não conseguiu implementá-las.

8.2 Principais métodos a implementar

Para se poder proceder à criação dos testes e à implementação e codificação da aplicação, primeiro tinham de se definir quais os principais métodos a ser implementados. Desta forma, o grupo conseguia ter uma noção do que é que teria de implementar:

- **enviarEstatisticas() -** Este método vai servir para mostrar todas as estatísticas aos clientes. Tem de enviar ao cliente todos os dados do radar, mostrar o histórico, mostrar as estatísticas de cada um dos sentidos. Vai à base de dados e retira de lá os dados, enviando-os ao cliente através da conexão Socket.
- **autenticarUser(**String user, String pass**) -** Este método tem de receber o user e a pass, para depois verificar se existe na bd. Se existir e coincidir retorna *true*, se não retorna *false*.
- **verificarAdmin(**String user, String tipo**)** - Este método tem de receber o user e o tipo de entidade, campo da Type_Entidade da tabela Entidades na base de dados. Depois vai verificar se é um administrador ou não. Retorna true se for administrador e false se não for.
- **registarEntidades(**String nome, String user, String mail, String type**) -** Este método fará o registo de novas entidades, inserindo-as na Base de Dados. Recebe os dados da entidade a registar. Cria uma pass gerada automaticamente. Coloca o user e a pass na tabela login da Base de Dados e coloca o nome, user, mail e tipo na tabela das entidades da BD. Retorna true se conseguir efetuar o registo.
- **exQuery(**String query**)** - Este método recebe uma string que será a query, envia-a à base de dados e não irá retornar nada
- **connect() –** Este método irá fazer a conexão com a BD, não tem parâmetros, mas vai retornar a ligação com a base de dados
- **configurarSistema() –** Este método iria servir para o administrador fazer a configuração do sistema, adicionar novos radares, etc. Contudo, esta funcionalidade era menos prioritária então o grupo decidiu-se focar antes nos outros métodos.

9 Implementação da aplicação no NetBeans

Para a implementação das classes e métodos, escolheu-se o NetBeans. O NetBeans IDE é um ambiente de desenvolvimento integrado gratuito e *open source* para programação usando Java. Escolheu-se este programa uma vez que é fácil e simples de usar. Além disso, como já tinha, anteriormente, sido usado nas aulas, o grupo já estava habituado a trabalhar neste programa, não sendo necessário aprender a usar um novo software.

9.1 Arquitetura da aplicação Cliente – Servidor

Neste projeto utilizou-se a arquitetura cliente-servidor, onde se criou uma aplicação que funciona como servidor, permitindo assim processar e enviar vários dados que são pedidos pelos clientes. A aplicação cliente é a mesma para todos os clientes e permite obter diversos dados do servidor. Implementou-se esta arquitetura usando Multithreading de forma a permitir que várias instâncias do cliente corram e recebam dados do servidor em simultâneo.

Quando a aplicação do servidor é aberta, as conexões são criadas e este fica à espera de aceitar clientes, utilizando comunicação por sockets. Além da conexão principal que serve para enviar os dados para o cliente, criou-se também uma conexão adicional que serve para enviar dados críticos para a autenticação. Ou seja, foi criado um servidor para o Login, sendo que este é iniciado a partir da aplicação do servidor principal.

Quando a aplicação do cliente é iniciada, a mesma é automaticamente conectada à socket do servidor. Este vai buscar todos os dados à base de dados e envia-os ao cliente, que os recebe de imediato e mostra-os na sua interface principal. Caso o cliente queira fazer login, ao abrir a janela de login do cliente, essa janela vai ser conectada à socket do servidor dedicado à autenticação. Uma vez feita a autenticação, o cliente terá acesso a funcionalidades adicionais, sendo que se for um administrador terá ainda mais particularidades.

Assim, para explicar melhor a estrutura deste projeto, foi feito um diagrama de componentes, apresentado na figura 6. Este diagrama captura a estrutura física da implementação e é construído como parte da especificação da arquitetura. Contém os componentes, interfaces e ficheiros e as relações entre eles.

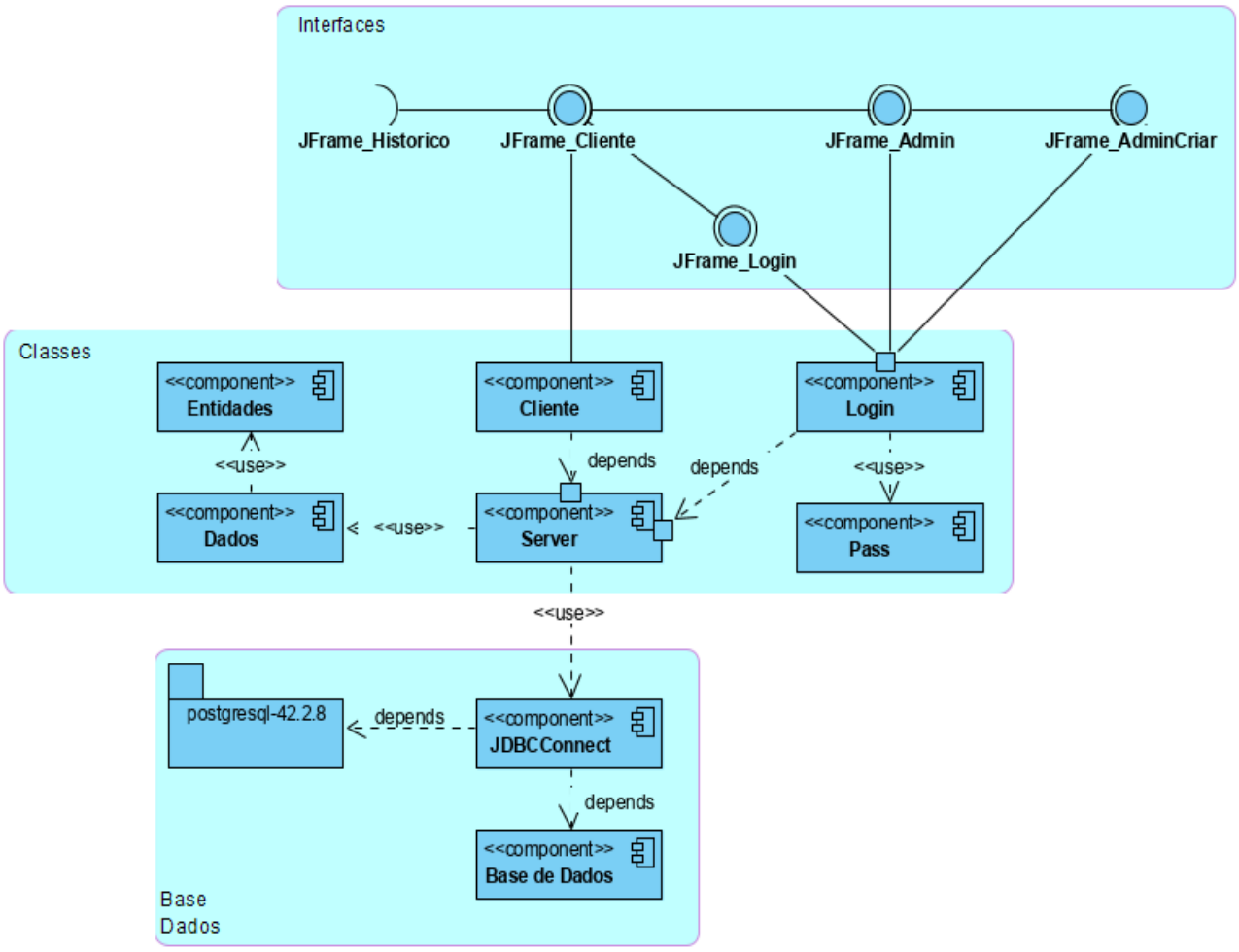

Figura 6 - Diagrama de Componentes

9.2 Implementação classes

As classes que foram realmente implementadas no NetBeans, acabaram por ser diferentes do que se tinha originalmente planeado. Algumas das classes não foram implementadas no NetBeans porque eram apenas classes de dados, ou seja, seriam implementadas na base de dados, mas não tinham necessariamente de ser criadas como classes java. Como foi o caso da classe Radar, a classe DadosObjetos e a classe Type. Estas foram apenas classes implementadas na base de dados. A classe Estatísticas, que serviria para gerar as várias estatísticas foi apenas implementada através de views na base de dados, tal como todos os seus métodos. Contudo, era na mesma necessário armazenar nalguma classe todos estes dados, de modo a poderem ser enviados para os clientes. Como tal, foi criada a classe **Dados**, onde eram guardados todos os dados que o servidor ia pedir à base de dados. Implementou-se também a classe **Entidades,** para armazenar os dados de todas as entidades para depois serem enviados ao cliente.

Relativamente às classes Administrador e Condutor, estas não foram implementadas. Existindo apenas uma classe **Client,** que seriam todos os utilizadores da aplicação. Acrescentou-se ainda a classe **Login,** para ser feita a autenticação dos clientes e a class **Pass**, para gerar passwords aleatórias. A classe **Server** e a classe **JDBCConnect** foram mantidas, sendo que apenas lhes foram implementados mais métodos.

Assim, de seguida na figura 7, pode-se verificar o diagrama de classes final deste projeto.

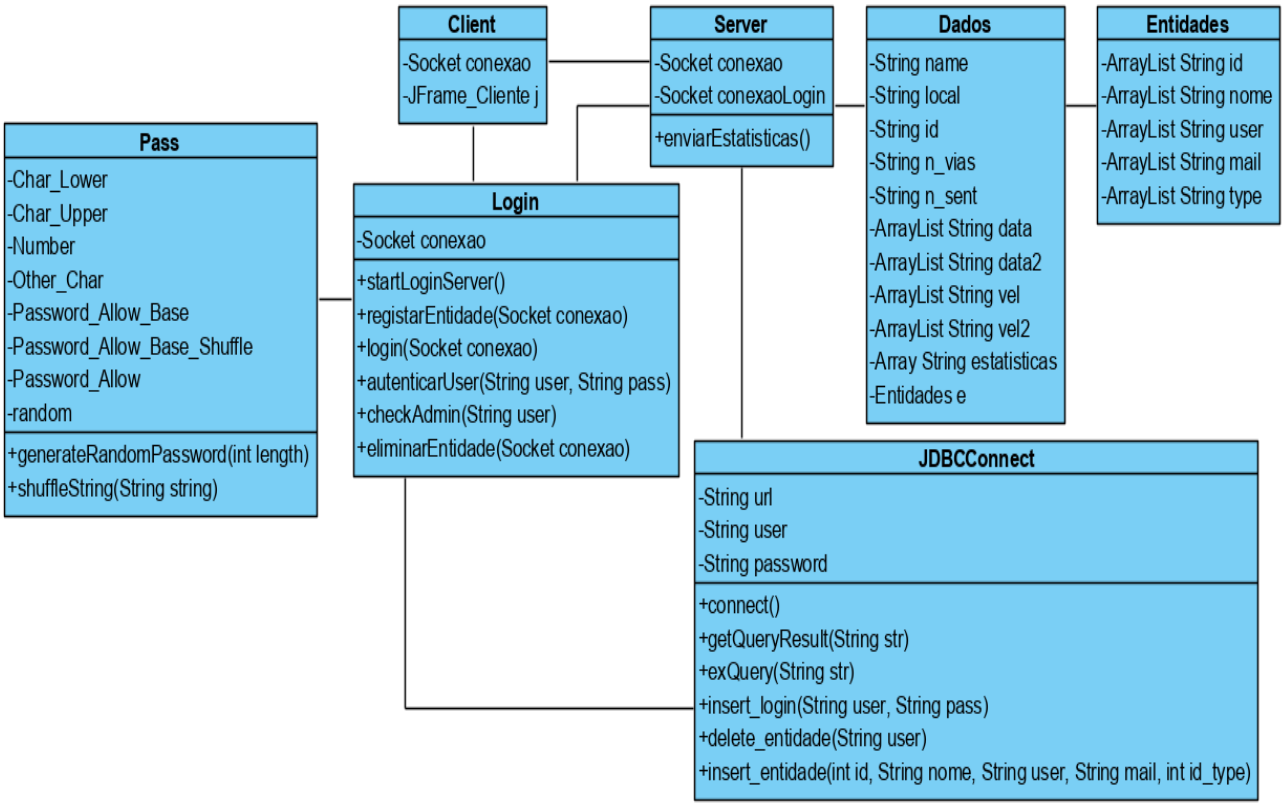

Figura 7 - Diagrama de Classes Real

9.2.1 Funcionalidades e métodos implementados

Este sistema, como já foi referido, funciona através de multithreading, com um modelo cliente-servidor. A classe Server é o servidor e este fica à espera dos clientes. Assim que um cliente se conecta, o servidor através do método, enviarEstatísicas (), vai à base de dados (através da classe JDBCConnect, que possui métodos para executar queries na base de dados), guarda todos os dados numa instância da classe Dados e envia esse objeto ao cliente. De seguida tendo os dados, as devidas páginas das interfaces iram mostrar os dados.

Assim, o método enviarEstatísticas () permite que todos os clientes vejam os dados do radar e as estatísticas de trânsito geradas. No caso das entidades, ou utilizadores registados, estas têm ainda acesso ao histórico, onde podem verificar todos os registos de velocidade guardados na base de dados de ambos os sentidos e também, a velocidade mínima e máxima.

As entidades são divididas em 2 tipos: administrador e não administrador. O administrador tem a particularidade de poder fazer a gestão das entidades, podendo registar e eliminar entidades.

Para os clientes fazerem a autenticação, estes conectam-se ao servidor de Login, que também tem acesso à base de dados, e através dos métodos autenticarUser() e checkAdmin() verifica se o username e password inseridos são válidos e se se trata ou não de um administrador.

No caso de o cliente ser um administrador, este pode registar uma nova entidade. Ele insere o username, nome e e-mail, que no caso de serem inválidos a aplicação envia um aviso (a password que é usada no login é criada automaticamente pelo servidor). Depois de validados os dados, o administrador conecta-se ao servidor de Login, que através dos métodos da classe JDBCConnect, insert_login() e insert_entidade(), é inserida uma nova entidade na base de dados. Para eliminar, o administrador seleciona uma entidade, conecta-se também ao servidor de login e através do método delete_entidade(), elimina o registo da base de dados.

Assim, foram implementadas as funcionalidades de mostrar estatísticas, mostrar o histórico, fazer autenticação e registar e eliminar entidades.

9.3 Conexão à base de dados – JDBC

Java Database Connectivity, ou JDBC, é uma interface de programação de aplicativos (API), que reúne conjuntos de classes e interfaces escritas na linguagem Java. Nesta linguagem, o JDBC possibilita a conexão, através de um driver específico, a uma base de dados desejada.

Com este driver pode-se executar instruções SQL do tipo “Select”, “Insert”, “Update”, ”Delete”, “Create” e “Drop” em qualquer tipo de base de dados relacional. Sendo depois possível invocar “Store Procedures”, que são funções ou procedimentos que ficam armazenados no Sistema de Gerenciamento de Banco de Dados ou SGBD escritos numa linguagem própria, através do JDBC. Além deste driver funcionar como uma interface entre os SGBD e as aplicações, também pode ser considerado como um tradutor que ajuda na definição das mensagens binárias trocadas com um SGBD.

Neste caso, para a base de dados foi usado o PostgresSQL, então para desenvolver uma aplicação utilizando o JDBC, usou-se o PostgreSQL JDBC 4.2 Driver, 42.2.9 e foi ainda necessário importar a biblioteca java.sql.

9.4 Criação interfaces em JFrame

Para a criação das interfaces da aplicação, o grupo baseou-se numas interfaces que tinham sido inicialmente desenhadas. Estas interfaces iniciais (figura 8) serviram apenas como guia, sendo que foram implementadas versões mais simples destas (figura 9).

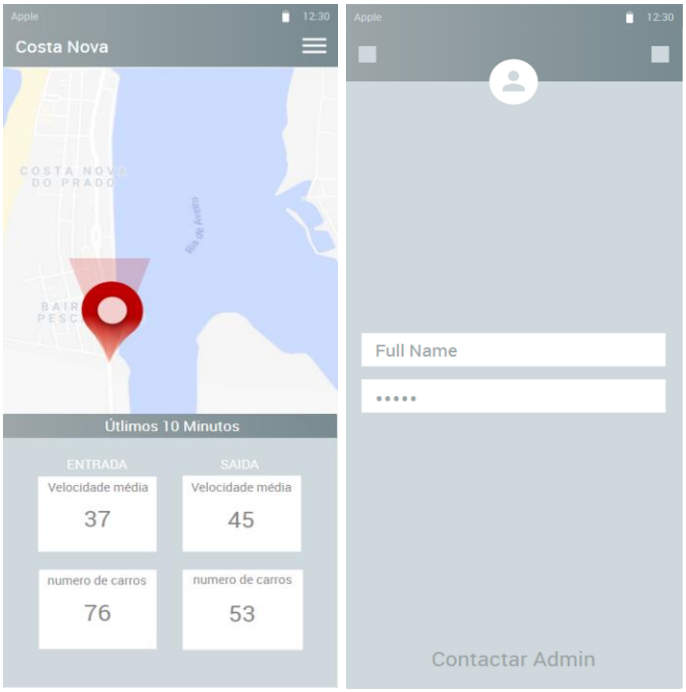

Figura 8 – Exemplo de interfaces inicialmente planeadas

As interfaces foram criadas usando o JFrame, pois é simples e o grupo sentiu-se mais à vontade para trabalhar com a ferramenta. O JFrame é uma ferramenta existente no JAVA, que permite criar interfaces simples e intuitivas.

O programa inicia-se no JFrame do Cliente, onde se podem ver os detalhes do radar, que contém o nome, a localização, o número de vias e número de sentido, e também as estatísticas referentes ao tráfego dos últimos 10 minutos, podendo-se alterar o seu sentido. Estas estatísticas

são: a contagem, que conta o número de veículos por sentido, a velocidade média, a velocidade máxima, velocidade mínima, e também uma classificação qualificativa do estado do trânsito.

Caso o utilizador seja uma entidade e tenha uma conta registada poderá efetuar o login. Para tal é necessário inserir um username e uma password válidas. A interface do cliente registado é igual à do sem registo mas tem acesso ao histórico de velocidades. No histórico podem-se visualizar todas as velocidades, incluindo a máxima e mínima de sempre podendo-se mudar de sentido ao clicar no Mudar Sentido.

No caso de ser um administrador, além de ter acesso a tudo que os clientes registados têm podem registar e apagar entidades. Na JFrame_Admin podem-se ver todas a entidades registadas, bem como seus dados excluindo a password, também se pode apagar entidades selecionando a que se pretende e carregando em apagar entidade e confirmar.

Na JFrame_Criar, o administrador pode registar novas entidades, sendo necessário um nome, username não registado, email válido e não registado, caso algo esteja mal o programa mostra uma mensagem de erro a indicar o que está errado.

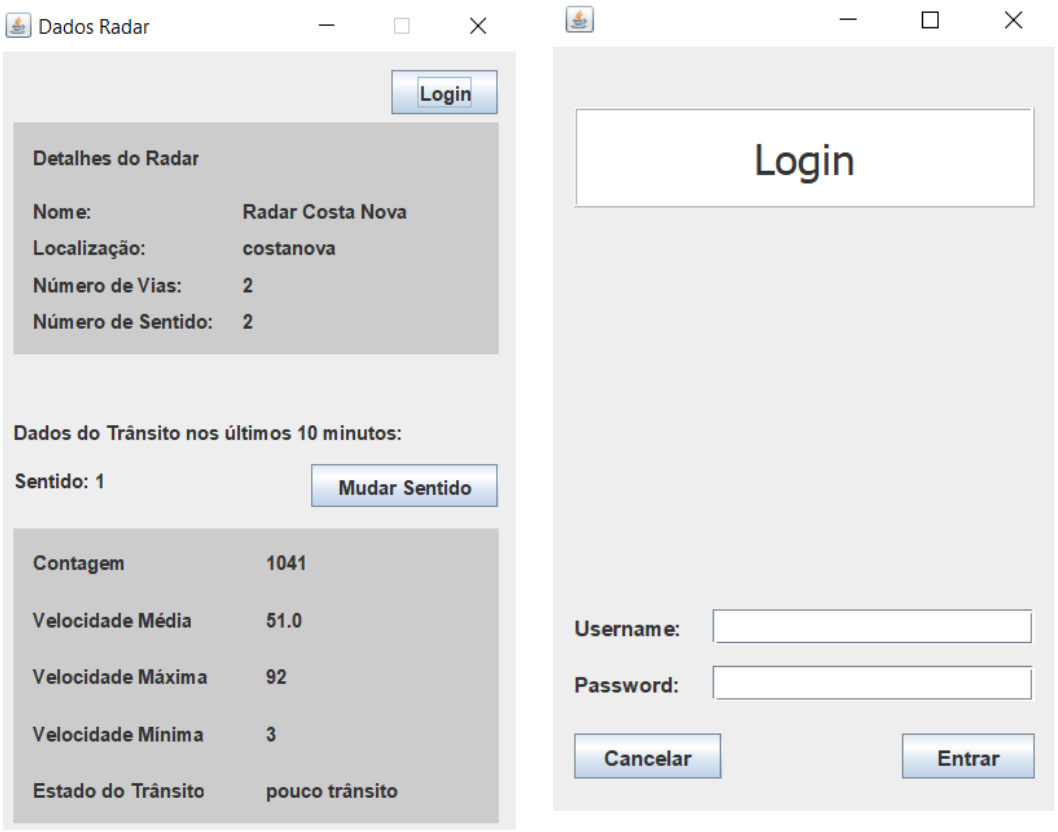

Figura 9 - Exemplo das interfaces criadas

10 Testes

10.1 Testes Unitários

Como já foi referido anteriormente, o grupo não conseguiu substituir a base de dados utilizando ferramentas como o JMock para realizar testes unitários. Como tal, alguns testes de JUnit foram feitos de forma diferente, de modo a não criar uma instância do objeto da classe a ser testada. Desta forma, foi possível utilizar os métodos dessa classe mais rapidamente e sem utilizar Threads.

Para manter a consistência dos testes, os métodos criados dentro da Classe de teste são semelhantes aos métodos originais de cada classe a ser testada, com a única diferença de usar dados fictícios para simular os dados reais da base de dados, através de Arrays.

10.2 Testes ao Sistema

Ao longo do desenvolvimento do programa, foram sendo feitos testes às interfaces, isto é, corria-se a classe Server e a classe Cliente e testava-se a aplicação. Clicava-se nos vários botões, experimentavam-se inserir vários inputs, corretos e incorretos e verificava-se se ocorria algum erro. Quando se detetavam os erros, estes eram analisados e tentava-se corrigi-los e repetia-se este processo.

A partir do momento em que foram testados com sucesso os requisitos funcionais e não funcionais, concluiu-se que a aplicação, apesar de algumas falhas, estava praticamente funcional.

Testou-se ainda qual era o número máximo de ligações a clientes que o servidor aguentava, chegando à conclusão que eram 100, uma vez que este é o limite máximo de ligações ao PostgreSQL.

11 Análise dos Resultados

Para fazer a análise dos resultados optou-se por identificar quais os requisitos que foram cumpridos, apresentando esses dados no formato de duas tabelas, uma para a análise dos requisitos funcionais (tabela 9) e outra para os requisitos não funcionais (tabela 10). Sendo que o símbolo "+" representa o cumprimento do requisito e o símbolo "-" representa o não cumprimento. O símbolo +/- significa que ficou incompleto.

Tabela 9 - Análise dos requisitos funcionais

Ref[a]	**Requisito funcional**	**Análise de Resultados**
RF.1	Permite a consulta das informações relativas às velocidades das viaturas em ambos os sentidos (histórico de velocidades).	+
RF.2	Permite a consulta da contagem das viaturas que passam pelo radar (tráfego).	+
RF.3	Permite a consulta da velocidade média em intervalos de 10 em 10 minutos.	-
RF.4	Sistema indica a velocidade máxima e mínima.	+
RF.5	Os condutores (sem registo) só terão acesso à velocidade média de trânsito (de 10 em 10 min) e ao contador de viaturas.	+/-
RF.6	As entidades registadas (Governo, IMT, Brisa…), terão acesso a todas as estatísticas disponíveis.	+
RF.7	Permitir a configuração da aplicação, de modo a funcionar para qualquer radar, independentemente da sua localização.	-
RF.8	Permitir o registo e eliminação de entidades.	+

Tabela 10 - Análise de Requisitos de Interface e Usabilidade

Ref[a]	Requisito de interface e usabilidade	Análise de Resultados
RInt.1	A interface do condutor é a interface principal, sendo logo mostrada quando se abre a aplicação.	+
RInt.2	A interface do condutor possui um mapa com a localização do radar, tendo também indicação do número de vias e sentidos nessa zona.	+/-
RInt.3	Mapa tem de ser manipulável, mas será possível afastar-se muito da zona do radar.	-
RInt.4	A interface do condutor possui também uma tabela com as informações da velocidade média para cada um dos sentidos, a informação da contagem dos carros, mostrando apenas a diferença e a avaliação do trânsito (muito trânsito, trânsito moderado e pouco trânsito).	+
RInt.5	A interface das entidades possui uma tabela com as velocidades máxima e mínima e uma tabela com um histórico de todas as velocidades.	+
RInt.6	O administrador tem ainda acesso a uma interface de configuração onde faz o registo das novas entidades e as outras configurações da aplicação.	+/-

12 Reflexão Crítica e Conclusão

No geral, foram cumpridos a maioria dos objetivos e dos requisitos definidos inicialmente, apesar de certas funcionalidades que tinham sido inicialmente planeadas terem sido descartadas, com o intuito do grupo se focar no que realmente era importante. Definiu-se o que tinha a maior prioridade e foi isso que foi implementado.

Em relação ao planeamento do projeto, como se optou por uma abordagem ágil, o desenvolvimento do projeto deveria ser dirigido por testes. Contudo, durante as fases iniciais, o grupo teve algumas dificuldades no planeamento, o que gerou alguma confusão para os membros do grupo. Além de que o planeamento do projeto levou mais tempo do que deveria.

Assim, como o grupo demorou um pouco mais do que devia a perceber o que tinha de ser implementado e como, acabou também por sofrer um pequeno atraso no desenvolvimento do projeto. Mesmo assim, o grupo conseguiu ultrapassar essas dificuldades, definir prioridades, refinar os diagramas e seguir em frente.

Quando se chegou à fase da criação dos casos de testes, estes não foram bem definidos, não sendo logo feitos no JUnit. Definiram-se quais os principais métodos que tinham de ser implementados no NetBeans, quais os seus parâmetros e o que tinham de retornar. Essa definição foi o suficiente para dar ao grupo uma noção do que tinha de ser implementado, passando então para a parte da programação. Posteriormente, tentou-se criar os testes no JUnit, contudo o grupo teve muitos problemas na criação de testes sem usar a base de dados. Foram procuradas várias soluções mas nenhuma delas foi implementada com sucesso. Como tal, tentou-se ultrapassar esta situação criando Arrays com dados fictícios para servirem de tabelas de bases de dados e modificaram-se os métodos de forma a testá-los sem conectar com a base de dados.

Relativamente à criação da Base de Dados, esta não gerou muitos problemas, mas mesmo assim, o código foi na mesma refactorizado várias vezes, pois iam sendo encontrados alguns problemas na criação das tabelas e das views.

A implementação da aplicação no NetBeans, foi definitivamente das partes mais complicadas do projeto. A implementação das várias classes foi sendo feita por partes e demorou algum tempo até tudo estar em ordem.

No que toca aos objetivos deste projeto, planeava-se conceber uma aplicação que:

1. permitisse a gestão do tráfego para qualquer zona à escolha do administrador;

2. permitisse a qualquer condutor obter estatísticas do tráfego;
3. permitisse a certas entidades o acesso a informações mais detalhadas do trânsito.

Os objetivos 2 e 3 eram os que tinham prioridade e foram cumpridos. Relativamente ao objetivo 1, este não foi cumprido, uma vez que esta aplicação só permite a gestão de apenas um radar e o administrador não consegue configurar o sistema adicionando novos radares.

Em relação à análise dos resultados que foi apresentada anteriormente, dos requisitos funcionais não foi cumprido o RF.7, pois este era o que estava relacionado com a configuração da aplicação por parte do administrador. O RF.3 e o RF.5 também não foram cumpridos na sua totalidade, pois referiam-se à consulta da velocidade média em intervalos de 10 em 10 minutos. Como não foi implementado na base de dados a receção de dados contínua, não se podem gerar dados de 10 em 10 minutos, não sendo possível verificar se a aplicação funcionaria nesse caso, deixando estes requisitos incompletos.

Relativamente aos requisitos de interface e usabilidade, simplificaram-se as interfaces do utilizador, inicialmente pensou-se em ter um mapa que seria manipulável, mas optou-se pela criação de interfaces mais simples apenas mostrando as estatísticas. Assim, não foi cumprido o RInt3, que refere que se deveria ter um mapa manipulável, e o RInt.2 ficou incompleto, pois refere que a interface do condutor deveria possuir o tal mapa. O RInt.6 também ficou incompleto pois só se criaram as interfaces para o registo de entidades.

Em conclusão, a aplicação que foi criada tem algumas falhas e poderiam ser feitas algumas melhorias e otimizações, mas mesmo assim, apesar das dificuldades e de não ter corrido tudo como estava planeado, foi possível criar uma aplicação que cumpria a maioria dos requisitos e objetivos.

13 Fontes e material de referência

Documentação PostgreSQL - https://www.postgresql.org/docs/

Documentação pgAdmin - https://www.pgadmin.org/docs/

Documentação Java - https://docs.oracle.com/javase/8/docs/

Documentação Netbeans - https://netbeans.org/kb/

Java Socket Programming - https://www.javatpoint.com/socket-programming

Documentação Junit - https://junit.org/junit4/

Jmock tutorial - http://jmock.org/getting-started.html

Elearning - https://elearning.ua.pt/?lang=pt

Geeks for Geeks - https://www.geeksforgeeks.org/

StackOverflow - https://stackoverflow.com/

Anexo A – Planeamento:

Glossário:

Aplicação - ou app é um programa informático que visa facilitar a realização de uma tarefa num computador ou num dispositivo móvel.

Atributo - característica de objeto em programação orientado a objetos.

Base de dados - Conjunto de dados organizados e relacionados, capaz de ser processado por um sistema informático.

Chave estrangeira - designação dada a um campo de uma tabela de uma base de dados referente à chave primária de uma outra tabela.

Chave primaria - designação dada a um campo de uma tabela de uma base de dados, cujos valores têm de ser sempre únicos, de modo a referir-se a apenas um registo da tabela.

Classe - um modelo de código de programa extensível para criar objetos, fornecendo valores iniciais para certas variáveis e funções ou métodos.

Dados brutos - dados não processados ou organizados.

Integer - número inteiro.

Interface - elemento que proporciona uma ligação física ou lógica entre dois sistemas ou partes de um sistema que não poderiam ser conectados diretamente.

Iterações - é o processo de repetição de uma ou mais ações.

Data stream - uma continua transmissão de dados.

Método - código de um programa que realiza uma certa tarefa

Modelo cliente-servidor - modelo de comunicação em que vários clientes comunicam com um servidor para receber vários tipos de informação

Multithreading - característica de um sistema operacional do computador que permite executar várias partes de um programa simultaneamente ou em rápida sucessão.

Open source - normalmente um código que pode ser utilizado e alterado sem restrições.

Query - comando que uma base de dados recebe para fazer certas pesquisas.

Refactorizar - processo de revisão de código com o objetivo de o tornar mais eficiente.

Sockets - elo entre os processos de um servidor e um cliente.

SQL - linguagem de programação que é utilizado em bases de dados relacionais.

Varchar - caracter varying, termo utilizado para designar um tipo de dados que é texto.