



INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CAMPUS TERESINA CENTRAL
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LUAN MACHADO PONTES

**STARBUS - UMA PLATAFORMA DE DADOS SOBRE O TRANSPORTE PÚBLICO
DE TERESINA**

TERESINA, PIAUÍ

2021

LUAN MACHADO PONTES

STARBUS - UMA PLATAFORMA DE DADOS SOBRE O TRANSPORTE PÚBLICO DE
TERESINA

Relatório Técnico de software apresentado à Banca Examinadora como requisito para aprovação na disciplina de Trabalho de Conclusão de Curso II do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí.

Orientador: Prof. Dr. Adalton Sena

TERESINA, PIAUÍ

2021

LUAN MACHADO PONTES

STARBUS - UMA PLATAFORMA DE DADOS SOBRE O TRANSPORTE PÚBLICO DE
TERESINA

Relatório Técnico de software apresentado à Banca Examinadora como requisito para aprovação na disciplina de Trabalho de Conclusão de Curso II do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia do Piauí.

Aprovado pela banca examinadora em _____.

BANCA EXAMINADORA:

Prof. Dr. Adalton Sena

Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. M^º. TBD

Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

Prof. M^º. TBD

Instituto Federal de Educação, Ciência e Tecnologia do Piauí - IFPI

TERESINA, PIAUÍ

2021

LISTA DE ILUSTRAÇÕES

Figura 1 – Diagrama UML de Casos de Uso	10
Figura 2 – Diagrama UML de Casos de Uso	12
Figura 3 – Arquitetura do Sistema	13
Figura 4 – Diagrama entidade-relacionamento	14
Figura 5 – Implantação	18

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
REST	Representational State Transfer
UML	Unified Modeling Language
JSON	JavaScript Object Notation

SUMÁRIO

1	INTRODUÇÃO	6
1.1	Objetivos	7
1.1.1	Objetivo Geral	7
1.1.2	Objetivos Específicos	7
2	TECNOLOGIAS ENVOLVIDAS	8
3	MODELAGEM DO PROJETO	9
3.1	Levantamento de Requisitos	9
3.1.1	Requisitos Funcionais	9
3.1.2	Requisitos Não Funcionais	9
3.2	Diagramas de casos de uso	10
3.3	Diagramas de classe	11
3.4	Arquitetura do Sistema	13
3.5	Diagrama de Entidades-Relacionamentos	14
3.6	API REST	14
3.6.1	/auth	15
3.6.2	/applications	15
3.6.3	/users	16
3.6.4	/lines	16
3.6.5	/vehicles	16
3.6.6	/stops	16
3.6.7	/interations	17
4	SOFTWARE	18
4.1	Implantação	18
4.2	Testes	19
4.2.1	Testes Unitários	19
4.2.2	Testes Funcionais	19
5	CONSIDERAÇÕES FINAIS	20
	REFERÊNCIAS	21

1 INTRODUÇÃO

Após a Segunda Guerra Mundial, os processos de urbanização e industrialização do Brasil, levaram milhões de pessoas do campo à cidade. Segundo o IBGE, nos anos 60, mais da metade da população brasileira era considerada rural, após 50 anos 84,4% da população brasileira reside em áreas urbanas [1]. Os investimentos em infraestruturas não acompanharam o ritmo acelerado da mudança, o que levou ao agravamento dos problemas urbanos, incluindo os relacionados à mobilidade e mais diretamente o transporte público.

Diversos problemas afetam os passageiros em seu dia a dia, a superlotação provavelmente é o que mais interfere na sua qualidade de vida. Algumas empresas de ônibus, a fim de solucionar tal problema, disponibilizam diversos veículos na mesma linha em um curto intervalo de tempo durante os horários de pico, mas a ansiedade do usuários de chegar em casa, a instabilidade do horário dos ônibus e a incerteza se o ônibus extra já passou ou não, faz com que os passageiros normalmente peguem o primeiro veículo disponível, causando o mau aproveitamento da frota.

A espera de ônibus faz os usuário se exporem muitas vezes a uma situação de risco e desconforto. Cada minuto em um ponto de ônibus o passageiro fica exposto a intempéries do tempo, como chuva e sol e não raro, assaltos são registrados em tais locais. Muitas vezes o usuário obriga-se a estar na parada muito tempo antes do ônibus passar, por vezes, deixando de fazer suas atividades, perdendo sono ou deixando de passar mais tempo com a família.

Os problemas mencionados acima poderiam ser solucionados com o aumento do número de veículos. No entanto tal solução pode ser economicamente inviável, por outro lado oferecer informação de qualidade pode diminuir tempo de espera na parada de ônibus bem como desencorajar que um passageiro entre em um veículo lotado sabendo que outro da mesma linha está na iminência de passar, levando assim a um melhor aproveitamento da frota disponível.

Do ponto de vista das autoridades que cuidam do sistema como um todo, existe a dificuldade de levantar problemas em pontos específicos do sistema. Por exemplo, como o órgão responsável pelos os pontos de ônibus da cidade pode saber se os tais atendem as expectativas dos usuários? Ou como saber a visão do usuário a respeito da frota de veículos de uma determinada linha ou empresa? Atualmente cada órgão possui seu canal de contato com o usuário, que poderiam colher tais informações, mas tais canais requerem um esforço considerável do usuário, que pode desestimular o engajamento e a perda do sentimento que levaria ele a apontar problemas. Por outro lado as atuais plataformas digitais permitem o aproveitamento do sentimento de engajamento de forma imediata e com um esforço consideravelmente menor.

O presente trabalho descreve o desenvolvimento de uma plataforma que ofereça informações sobre o transporte público a desenvolvedores de aplicativos móveis, além de oferecer um repositório centralizado de avaliações realizadas por usuário sobre diversos aspectos do

transporte público. Com tal ferramenta os usuários poderão ter acesso fácil a informações sobre os horários, itinerários, localização de veículos e linhas por paradas, que podem ajudar a resolver os problemas citados anteriormente. Por outro lado, os mesmos poderão conceber uma visão sobre sistema transporte público local, por avaliarem pontos de ônibus ou estado de conservação dos veículos, informações essas que podem ser úteis as entidades que cuidam do transporte público da cidade.

1.1 OBJETIVOS

Abaixo são listados os objetivos gerais e específicos do presente trabalho.

1.1.1 Objetivo Geral

O objectivo geral deste trabalho é descrever o desenvolvimento de uma plataforma que centralize os dados do transporte público de Teresina e que facilite o desenvolvimento de novas aplicações voltadas para a melhoria da mobilidade urbana.

1.1.2 Objetivos Específicos

- Descrever os processo de desenvolvimento de uma REST API web, incluindo as decisões técnicas tomadas para a mesma.
- Demonstrar o funcionamento da plataforma criada.
- Expor a documentação da API implementada.

2 TECNOLOGIAS ENVOLVIDAS

As tecnologias utilizadas na construção da plataforma Starbus são divididos em três grupos de ferramentas. Abaixo é especificado cada grupo, bem como as ferramentas pertencentes a cada um deles.

O primeiro grupo abrange as ferramentas utilizadas para a construção da plataforma em si, sendo a linguagem de programação, o framework web e o banco de dados utilizado na aplicação. A linguagem Ruby [2] na versão 2.5 é uma linguagem de curta curva de aprendizagem, com abordagem pragmática para desenvolvimento de software e dispõe de um conjunto de ferramentas maduras que encorajam o desenvolvimento ágil. Faz parte desse ferramental o framework Grape, que tem se mostrado uma solução prática para a construção de uma API REST, focando em oferecer uma interface simples para manipulação de dados oriundos de requisições web. Por esse motivo, Grape [3] foi utilizado na concepção da API REST da plataforma. O dados são armazenados em banco de dados Postgres [4], na versão 12.1, que é de fácil configuração e amplamente adotado no mercado.

O segundo grupo é composto por ferramentas de suporte ao desenvolvimento. A primeira escolha foi o editor Vim [5] que possui uma excelente suporte a linguagem Ruby, além de permitir uma fácil alternância entre o terminal e o editor, tornando prática a execução de tarefas no terminal, que é prática comum no desenvolvimento Ruby. Para escrita dos testes da aplicação foi escolhida a biblioteca Rspec [6] na versão 3.9, que tem influenciado toda a comunidade de desenvolvimento de software com sua expressividade e clareza. Para acesso, debug e manipulação de dados no Postgres, foi utilizada a IDE DBeaver [7], na versão 7.0, um software open source com funcionalidades excelentes. Todo o controle do código foi feito por meio da ferramenta de controle de versão Git [8] na versão 2.25.1.

Por fim, para executar a plataforma Starbus em ambiente produtivo e assim atingir os objetivos específicos do trabalho, procurou-se uma ferramenta que pudesse trazer a menor carga de trabalho relacionada a infra estrutura de aplicação, visto que o foco do trabalho deveria ser no desenvolvimento da solução. Assim sendo, a plataforma Heroku [9] foi a solução mais viável, visto que oferece um ambiente extremamente flexível e escalável com apenas poucas linha de configuração. Por exemplo, para se colocar em execução uma nova versão da aplicação basta realizar um git push para um determinado repositório da ferramenta. Nesse terceiro grupo também inclui ferramentas que auxiliam na implantação e manutenção da qualidade do sistema e serão detalhadas na seção de implantação do software.

3 MODELAGEM DO PROJETO

3.1 LEVANTAMENTO DE REQUISITOS

O levantamento de requisitos foi feito a partir de conversas com um conjunto de desenvolvedores, que sugeriram as funcionalidades que poderiam ser interessantes para a plataforma, tendo em vista a sua proposta plataforma e as necessidades de desenvolvimento de aplicações. Segue listados os requisitos levantados.

3.1.1 Requisitos Funcionais

Abaixo são listados os requisitos funcionais do sistema:

1. **Fornecimento de localização de veículos:** Fornecer informações de localização dos veículos do sistema no momento solicitado.
2. **Catálogo de linhas disponíveis no Sistema:** Listar as linhas disponíveis no sistema.
3. **Catálogo das paradas de ônibus:** Listar as paradas disponíveis no sistema e as linhas que estão disponíveis em cada uma delas.
4. **Avaliação de entidades do sistema:** Registrar avaliações dos usuários sobre paradas e veículos do sistema de transporte público.
5. **Gerenciamento de usuários e autenticação:** Possibilidade de cadastro, edição e suspensão de usuários na plataforma.

3.1.2 Requisitos Não Funcionais

Abaixo são listados os requisitos não funcionais do sistema:

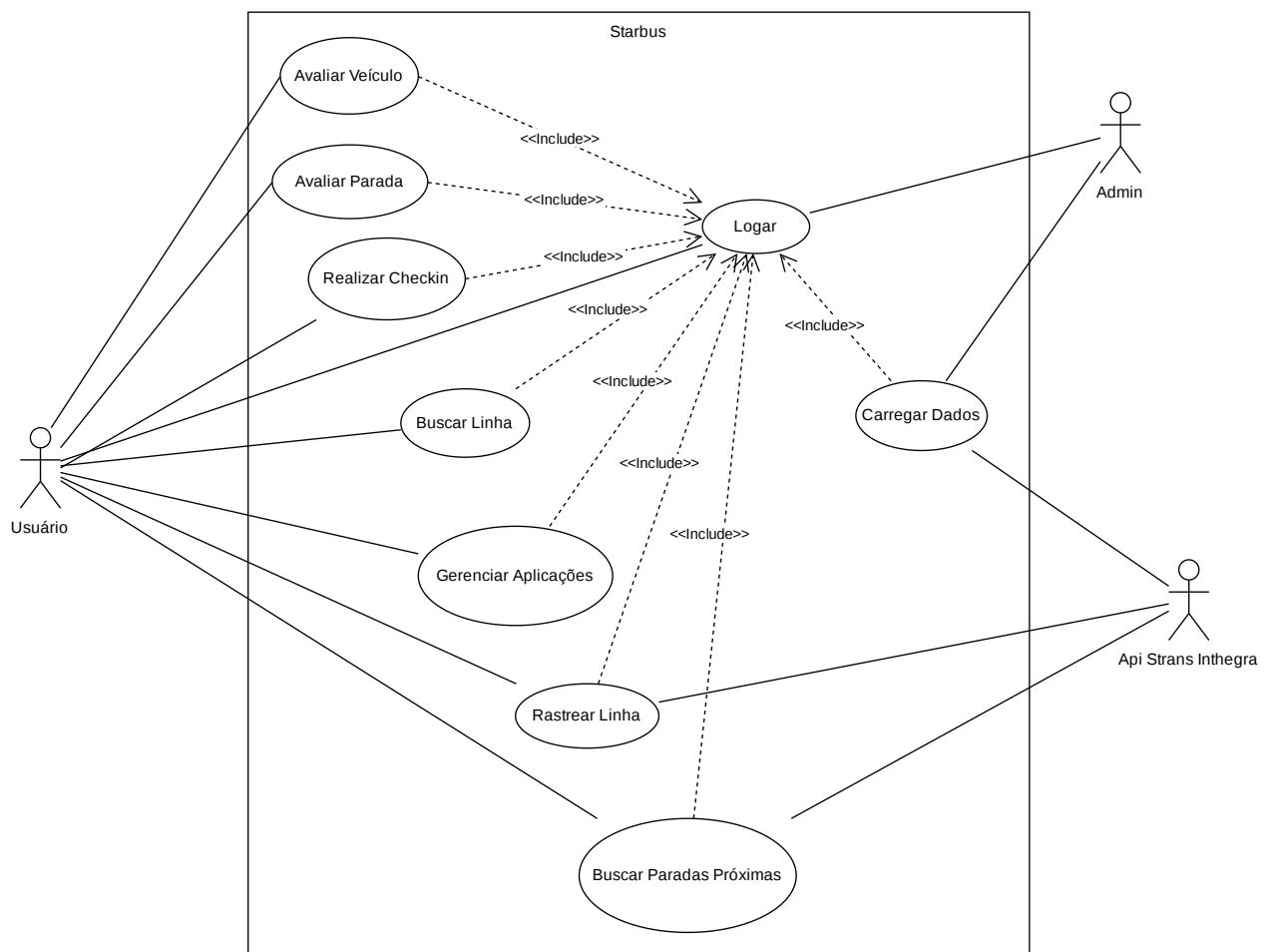
1. **Escalável e de alta disponibilidade:** Deve ser facilmente escalável para atender o aumento repentino de demanda ao mesmo tempo que mantém-se sempre disponível.
2. **Baixa carga de trabalho de Infra:** Baixa carga de trabalho de infra pode ajudar na evolução da plataforma, uma vez que o foco deve seguir no desenvolvimento de funcionalidades.
3. **API deve se simples e flexível:** Para ser atrativa ao uso de desenvolvedores a API deve ser bem documentada, prática e flexível.

3.2 DIAGRAMAS DE CASOS DE USO

O diagrama de caso de uso mostrado na figura 1 descreve as interações possíveis entre usuários e ferramentas.

Figura 1 – Diagrama UML de Casos de Uso

Visual Paradigm Online Diagrams Express Edition



Visual Paradigm Online Diagrams Express Edition

Caso de uso Starbus

O diagrama acima expressa os seguintes casos de uso:

1. **Logar:** Ação de identificar-se como usuário da plataforma que permite a utilização dos recursos da plataforma representados pelos casos de uso seguintes. Para manter a credibilidade das informações fornecidas pelo usuários é essencial a identificação de cada fornecedor.

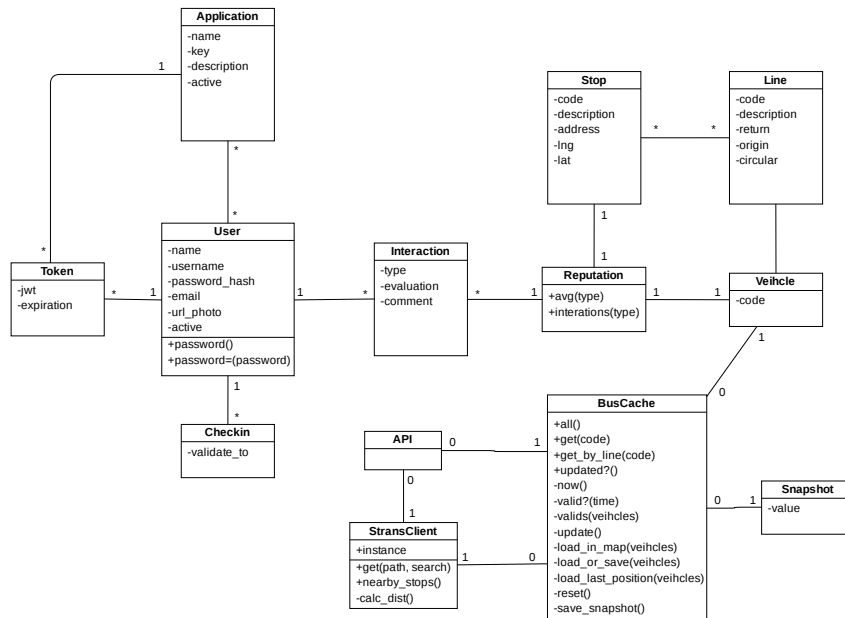
2. **Carregar Dados:** Usuário administrador pode carregar as informações de linhas e paradas a partir da API Strans Inthebra. Essa ação será executada sempre que houver necessidade de atualizar a base do Starbus com novas informações da API Strans Inthebra.
3. **Avaliar Veículo:** Usuário devidamente identificado publica sua opinião sobre um veículo, em aspectos como acessibilidade, estado de conservação, lotação e conforto;
4. **Avaliar Parada:** Usuário devidamente identificado publica sua opinião sobre uma parada de ônibus em aspectos como acessibilidade, estado de conservação, movimentação e conforto;
5. **Realizar Checkin:** Usuário devidamente identificado indica que no momento encontra-se em uma parada ou veículo específico; Essa informação é útil para levantar a lotação dos veículos e das paradas de ônibus.
6. **Gerenciar Aplicações:** Um usuário pode ter várias aplicações, que correspondem a diversos aplicativos que comunicam-se com a plataforma, cada um deles com seu conjunto de usuários associados. Aqui é possível gerenciá-las (ativar e desativar) ou criar novas aplicações;
7. **Buscar Linha:** Usuário fornece um código ou um termo para pesquisa que é usado para encontrar uma linha que ele deseja detalhes, como itinerário e localização dos veículos;
8. **Rastrear Linha:** Tendo um identificador de uma linha, que pode ser recuperado no caso de uso anterior, é possível consultar a localização dos veículos dessa linha em tempo real com dados fornecidos pela Api Strans Inthebra;
9. **Buscar Paradas Próximas:** Usuário fornece uma localização, que pode ser a sua atual ou de um ponto específico que queira chegar, a plataforma retorna todas as paradas próximas aquele ponto, com os dados da parada inclusive as avaliações dos usuários;

3.3 DIAGRAMAS DE CLASSE

O modelo de dados da aplicação apresentado na Figura 2, por meio de um diagrama de classes da UML.

Figura 2 – Diagrama UML de Casos de Uso

Visual Paradigm Online Diagrams Express Edition



Visual Paradigm Online Diagrams Express Edition

- **Application:** Representação de uma entidade que gera ou consome dados do Starbus, sendo que esta pertence a um usuário e pode ter diversos usuários associados a ela.
- **User:** A identidade de um usuário do sistema que acessa a plataforma por meio de uma aplicação. Todo usuário possui username e password que são utilizados para a autenticação no sistema, somente após a autenticação é possível que um usuário possa consumir ou produzir informações para a plataforma.
- **Token:** Identidade temporária para que um usuário logado possa interagir com a plataforma. O token é gerado no momento do login e armazenado para ser validado posteriormente a cada iteração com a plataforma. Um Token tem um ciclo de vida bem definido e curto.
- **Checkin:** Um usuário pode notificar a sua localização em um determinado instante, pode ser uma parada de ônibus ou em um veículo específico, tal informação poderá ser utilizado pela plataforma para verificar as condições de lotação das paradas ou dos veículos em determinado momento. O Checkin representa essa informação e está diretamente relacionado ao caso de uso cinco, Realizar Chekin.
- **Interaction:** Dados e comportamento dos casos de uso de Avaliar Veículo e Avaliar Parada, que são sumarizados, na classe seguinte Reputation;
- **Reputation:** Sumarização de todas as avaliações realizadas por usuários a uma entidade do sistema de transporte público, representando a reputação de cada veículo

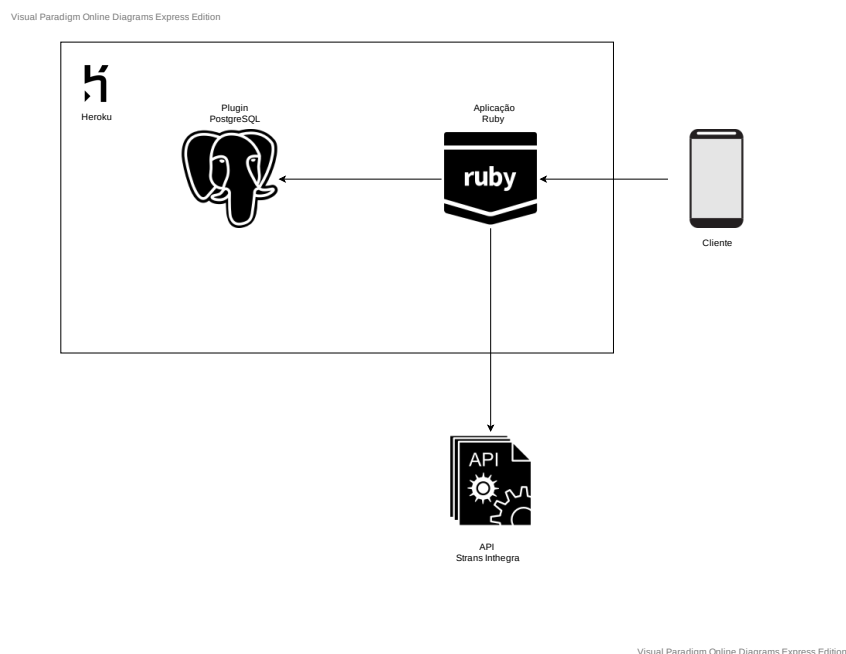
ou parada do sistema a ela associada.

- **Stop:** Parada de ônibus, com seu identificador, endereço e localização.
- **Line:** Linha de ônibus, com seu identificador e itinerário.
- **Vehicle:** Cada veículo de transporte de passageiros.
- **BusCache:** Otimização que permite diminuir o número de consultas a API Strans, armazenamento em memória todas as consultas realizadas a api referente a localização de veículos. Guarda as regras para a definição de dados desatualizados, que permitem uma nova consulta a API.
- **Snapshot:** Cada vez que uma nova consulta de localização de veículos é realizada a API um JSON é armazenado para conservação do histórico da localização de veículos em diferentes horários do dia, futuramente esse dados podem ser usados para estudar o comportamento do sistema e sugerir melhorias.
- **StransClient:** Interface simplificada de consulta a API Stran Inthebra.

3.4 ARQUITETURA DO SISTEMA

A arquitetura do sistema é apresentado na Figura 3 onde são apresentados o componentes do sistema e sua respectivas descrições abaixo.

Figura 3 – Arquitetura do Sistema



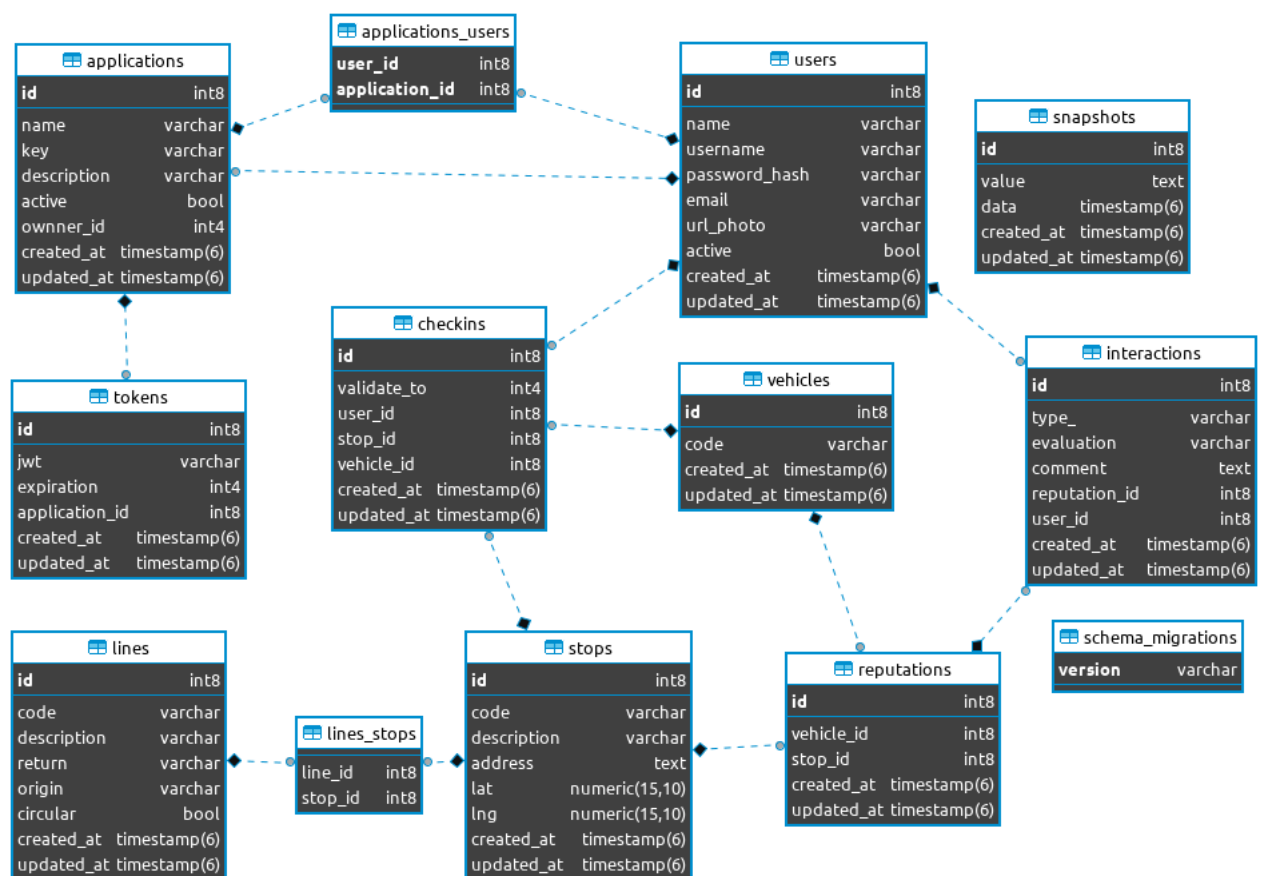
- **Client:** Um cliente da plataforma que pode consumir ou publicar dados por meio de uma aplicação.
- **Plugin Postgres:** Um plugin da plataforma Heroku que provisiona uma instância de Postgres configurada e altamente escalável de forma rápida e prática.

- **API Strans Inthebra:** API disponibilizada pela Strans, que fornece informações sobre o transporte público de Teresina.
- **Ruby Application:** Uma aplicação Ruby, responsável em receber requisições de clientes, realizar consulta e armazenar no banco de dados e recuperar informações da API Strans Inthebra, tudo de maneira estruturada obedecendo a uma API REST.

3.5 DIAGRAMA DE ENTIDADES-RELACIONAMENTOS

Na figura 4 podemos ver a estrutura lógica usada como base para o banco de dados. A maioria representam as entidades detalhadas anteriormente no Diagrama de Classe, com exceção da entidade `schema_migrations`, que é utilizada pelo framework Active Record para armazenar o histórico de migrações utilizadas por ele.

Figura 4 – Diagrama entidade-relacionamento



3.6 API REST

Como já descrito, essa plataforma que utiliza o conceito de API REST e toda a comunicação é feita por meio de métodos http, que são organizados semanticamente com base em resources, abaixo serão descritos os resources bem como os métodos que são disponíveis na API

da plataforma. Toda a transmissão de dados será feita utilizando o padrão JSON, assim sendo o corpo das requisições bem como das respostas da API serão texto obedecendo as regras desse padrão amplamente difundido na WEB.

Todas as requisições à API devem ser feitas utilizando o cabeçalho de "Authorization" com o token de autenticação recuperado de "auth/login", intuitivamente essa chamada é a única que não é obrigatória a inclusão do cabeçalho de autenticação.

Por padrão os recursos abaixo retornam os seguintes códigos HTTP, para cada situação descrita.

- **200**: Solicitação executada com sucesso.
- **403**: Token inválido ou não enviado.
- **404**: O path não foi encontrado ou não existe entidade com id/código especificado.
- **415**: Solicitação fora do formato esperado pela API.
- **500**: Erro interno do servidor.

A URL base da API é <https://starbus-v2.herokuapp.com/v2/> que é seguida pelos recursos detalhados a seguir.

3.6.1 /auth

Responsável pelos dados de autenticação na plataforma. Basicamente retorna o token de acesso que deve ser utilizado nas próximas requisições, e retorna dados sobre o usuário autenticado com o token utilizado.

Método	Caminho	Descrição
POST	/login	Autentica o usuário e retorna o token que será utilizado posteriormente
GET	/session	Retorna os dados do usuário logado com token passado

Tabela 1 – Resource Auth

3.6.2 /applications

Gerenciamento de aplicações da plataforma que só pode ser feito pelo usuário que possui as aplicações. Assim cada usuário gerencia suas próprias aplicações.

Método	Caminho	Descrição
POST	/	Cria uma nova Application com os dados passados
GET	/	Retorna todas as Applications
PUT	/:id	Atualiza a Application com id especificado, com os dados passados
GET	/:id	Retorna a Application com o id especificado
DELETE	/:id	Desativa a Application com o id especificada

Tabela 2 – Resource Applications

3.6.3 /users

Gerenciamento de usuários.

Método	Caminho	Descrição
POST	/	Cria uma nova User com os dados passados
GET	/	Retorna todas as Users
PUT	/:id	Atualiza a User com id especificado, como os dados passados
GET	/:id	Retorna a User com o id especificado
DELETE	/:id	Desativa a User com o id especificado

Tabela 3 – Resource Users

3.6.4 /lines

As Linhas disponíveis na aplicação são fornecidas e gerenciadas pela API Strans, assim o Starbus não faz o gerenciamento dessas linhas, ele apenas carrega tais linhas de sua fonte. O resource abaixo carrega tais linhas e oferece algumas consultas as linhas na base.

Método	Caminho	Descrição
POST	/load	Carregas as linhas a partir da API Strans
GET	/	Retorna todas as Lines
GET	/?codigos[]=:codigo	Retorna as Lines com os códigos especificados
GET	/:codigo/vehicles/	Retorna os Vehicles com na rota da linha com o código passado

Tabela 4 – Resource Lines

3.6.5 /vehicles

Assim como as linhas a Lines, Veihcles não são gerenciados pelo Starbus, a cada consulta na Strans a plataforma se o Veihcle já está cadastrado em sua base, se não ela cadastra. Sendo assim, esse resource retorna simplesmente os Veihcles rodando no momento da requisição.

Método	Caminho	Descrição
GET	/	Retorna todas as Vehicles rodando no sistema nesse momento
GET	/:codigo	Retorna o Vehicle rodando no sistema com sua posição no momento
POST	/:codigo/checkin	Realiza o Checkin na Stop com o código passado

Tabela 5 – Resource Vehicles

3.6.6 /stops

Stops também não são gerenciados pela plataforma e são carregados por quando as Lines são carregadas, a cada Line carrega são automaticamente salvos as paradas associadas a ela.

Método	Caminho	Descrição
GET	/	Retorna todas as Stops do sistema
GET	/:codigo/lines	Retorna todas as Lines de uma Stop com o código especificado
GET	/closes/?lat=:lat&long=:long	Retorna todas as Stops próximas a localização passadas por meio de lat e long
POST	/:code/checkin	Realiza o Checkin na Stop com o código passado.

Tabela 6 – Resource Lines

3.6.7 /interactions

Esse resource o é possível registrar as avaliações dos usuários sobre dois componentes do sistema de transporte público, paradas de ônibus e veículos.

Método	Caminho	Descrição
GET	/:criterio/vehicle/:codigo	Retorna a User com o id especificado
GET	/:criterio/stop/:codigo	Retorna todas as Users
POST	/:criterio/vehicle/:codigo	Cria uma nova User com os dados passados
POST	/:criterio/stop/:codigo	Cria uma nova User com os dados passados

Tabela 7 – Resource Interactions

Abaixo são listados os critérios que são utilizados para avaliar essas entidades, que nos caminhos são representados pelo parâmetro "criterio".

- **accessibility**: Acessibilidade de acesso para pessoas com dificuldade de locomoção.
- **comfort**: Conforto para o usuário.
- **safety**: Segurança, tanto no aspecto de uso como de segurança pública.
- **state**: Estado de conservação.

Para cada um dos critérios a acima o usuário pode associar uma avaliação listada a abaixo para a entidade que está avaliando que é especificada pelo ":codigo" da entidade.

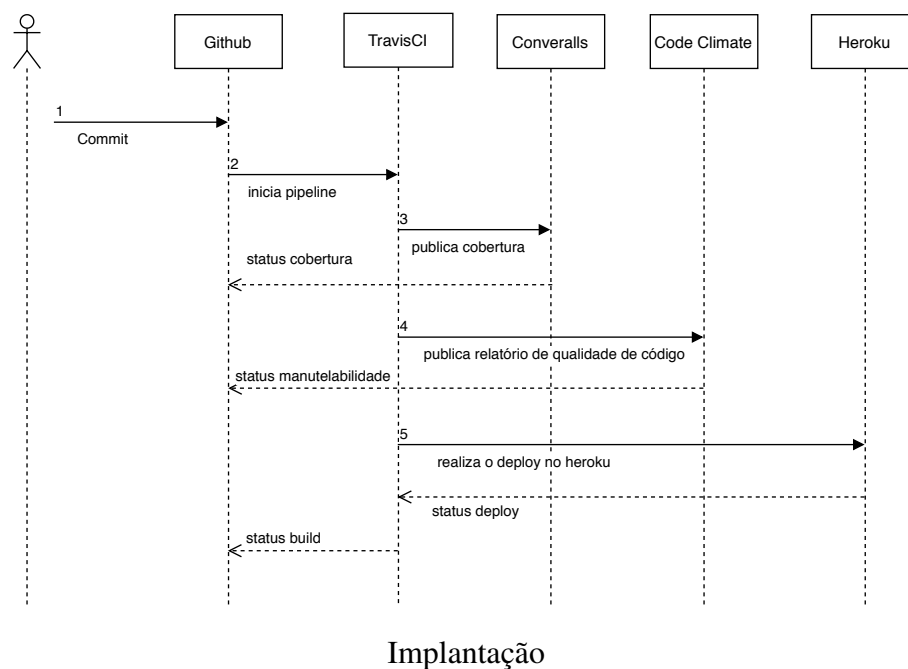
- **bad**: Ruim, longe de ser o ideal.
- **regular**: Regular, atende as necessidades do usuário.
- **good**: Muito bom, não deixando a desejar em nenhum aspecto.

4 SOFTWARE

4.1 IMPLANTAÇÃO

A plataforma utiliza-se do conceito de Continuous Delivery e Continuous Integration para garantir a entrega de software com agilidade e qualidade. O desenho abaixo mostra como funciona esse conceito na prática e todas as ferramentas envolvidas para a implantação do software.

Figura 5 – Implantação



As ferramentas envolvidas no processo e suas utilidades:

1. **Github:** Base de código que recebe o commit na branch master, sendo esse o gatilho que inicia o processo.
2. **TravisCi:** Ferramenta de integração contínua que imediatamente após o commit inicia uma pipeline de integração rodando todos os testes para aplicação. Para que a aplicação seja considerada apta para ambiente produtivo, todos os testes devem ser executados com sucesso e devem prover uma cobertura de testes superior a 90%.
3. **Coveralls:** Site de publicação de cobertura de testes, que guarda o histórico de cobertura por build.
4. **Code Climate:** Ferramenta de análise de código, nela é possível visualizar indicadores de manutenibilidade de código.

5. **Heroku:** Uma plataforma cloud que permite a realização de deploys de aplicações a partir de um git push em um repository específico.

4.2 TESTES

Testes tem um papel fundamental na manutenibilidade da plataforma Starbus, pois eles fornecem a segurança necessária para a evolução da base de código da plataforma. A cobertura de software pode ser considerado um requisito fundamental para se colocar um código em produção. Para realizar o levantamento da cobertura de código localmente, utilizou-se o parametrizável `simplecov`, uma biblioteca que permite a geração de relatórios de cobertura de código em diversos formatos, inclusive os aceitos pelo `coveralls`, ferramenta apresentada anteriormente.

Abaixo são listados os dois tipos de testes de software implementados durante a construção da plataforma.

4.2.1 Testes Unitários

Estes testes focam em unidades menores de código e desconsideram o contexto em que tais unidades são inseridas. Para a implantação destes tipos de testes foi utilizado o framework de teste `Rspec`, que oferece um conjunto completo de ferramentas para a aplicação de asserts e simulação de comportamentos.

4.2.2 Testes Funcionais

Testes que garantem que as unidades de código estão trabalhando juntas dentro de um contexto a fim de atender a um determinado requisito do software. Para esses tipos de testes foi utilizado a biblioteca de testes `Airborne`, que permite realizar a simulação de requisições http.

5 CONSIDERAÇÕES FINAIS

Esse trabalho descreveu a plataforma Starbus, uma ferramenta que busca centralizar as informações sobre o transporte público de Teresina, bem como os aspectos técnicos envolvidos no seu desenvolvimento. A plataforma já se encontra em funcionamento e tem se tornado uma alternativa para desenvolvedores que buscam informações sobre o transporte públicos local. A Aplicação Starbus, disponível na Play Store, é a principal parceira da plataforma, já possui mais de 10 mil instalações e tem ajudado usuários a melhor aproveitarem os recursos do transporte publico de Teresina e está disponível no endereço <<https://play.google.com/store/apps/details?id=com.starbus&hl=pt-BR>>.

O software descrito neste trabalho tem o código fonte aberto e está disponível no seguinte repositório no Github <<https://github.com/t00lmaker/starbus-v2>>, a sua documentação da API REST encontra-se <<https://documenter.getpostman.com/view/593922/Szf52UVe>>. Acesse o Starbus pelo link <<https://documenter.getpostman.com/view/593922/Szf52UVe>>.

REFERÊNCIAS

- 1 IBGE. *População nos Censos Demográficos, segundo as Grandes Regiões, as Unidades da Federação e a situação do domicílio - 1960/2010*. 2011. IBGE. Disponível em: <<https://censo2010.ibge.gov.br/sinopse/index.php?dados=8>>. Acesso em: 03 out. 2020. Citado na página 6.
- 2 LINGUAGEM de Programação Ruby. Disponível em: <<https://www.ruby-lang.org/pt/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 3 RUBY Grape | An opinionated framework for creating REST-like APIs in Ruby. Disponível em: <<http://www.ruby-grape.org/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 4 POSTGRESQL: The world's most advanced open source database. Disponível em: <<https://www.postgresql.org/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 5 WELCOME home : vim online. Disponível em: <<https://www.vim.org/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 6 SPEC: Behaviour Driven Development for Ruby. Disponível em: <<https://rspec.info/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 7 DBEAVER Community: Free Universal Database Tool. Disponível em: <<https://dbeaver.io/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 8 GIT. Disponível em: <<https://git-scm.com/>>. Acesso em: 03 out. 2020. Citado na página 8.
- 9 HEROKU | Cloud Application Platform. Disponível em: <<https://www.heroku.com/>>. Acesso em: 03 out. 2020. Citado na página 8.