MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

Bruna Do Espirito Santo Souza
Layane Grazielle Sousa Dias
Pedro Ivo Santana Melo

Arquitetura

SmartPark

Trabalho apresentado
na disciplina de Padrões
de Arquitetura de Software

Orientador: Dr. Jacson
Rodrigues Barbosa

Goiânia

1. Introdução

1.1 Finalidade

A principal finalidade deste documento é definir os aspectos essenciais da Arquitetura de Software, sendo direcionado aos stakeholders do projeto, possuindo grande foco para os Desenvolvedores e para a Equipe de implantação.

1.2 Escopo

O “SmartPark” é um estacionamento inteligente, que visa facilitar a busca por vagas de estacionamento, economizando tempo e reduzindo o tráfego desnecessário. O sistema fará isso através da identificação da chegada do motorista e da condução do mesmo até o estacionamento mais próximo, além de trazer uma visualização da situação do estacionamento para usuários administradores e a identificação através de sensores da ocupação de uma vaga.

1.3 Definições, Acrônimos e Abreviações

Id.: Identificador.
Software: Conjunto de documentações, guias, metodologias, processos, códigos e ferramentas para a solução de um problema.
Stakeholder: Indivíduo, grupo ou organização que possua interesse no Sistema.
Visão Arquitetural: Produto resultante da interpretação de um Stakeholder do sistema.
Arquitetura de Software: Forma como os componentes são agrupados com o objetivo de construir um software ou sistema.
Ponto de Vista Arquitetural: Produto resultante da execução de uma Visão Arquitetural.
Javascript: Linguagem de programação de alto nível, de propósito geral, interpretada, de sintaxe concisa e clara.
CRUD: Create(criar), Read (Ler), Update (Atualizar), Delete (Excluir).

1.4 Visão Geral

De maneira simples, o documento visa descrever a estrutura geral do sistema a ser desenvolvido, incluindo etapas de organização lógica e física, bem como de componentes e suas relações e ainda acerca de possíveis interfaces.

As principais decisões de projeto, restrições e metodologias adotadas para o desenvolvimento do projeto estão descritas e detalhadas. Além disso, vale ressaltar que esse Projeto Arquitetural se trata de um processo contínuo, sendo assim é suscetível a possíveis melhorias futuras.

2. Contexto da Arquitetura

2.1 Funcionalidades e Restrições Arquiteturais

Tipo	Id. do Documento de Requisitos
História de Usuário	HU-01
História de Usuário	HU-02
História de Usuário	HU-03
História de Usuário	HU-04
História de Usuário	HU-05
História de Usuário	HU-06
História de Usuário	HU-07
História de Usuário	HU-08
História de Usuário	HU-09
História de Usuário	HU-10
História de Usuário	HU-11

Requisitos não funcionais

RNF 01 - Usabilidade: o sistema deve ser intuitivo e fácil de usar para os usuários, com uma interface amigável que permita uma experiência simples e eficiente durante a entrada, saída e busca por vagas de estacionamento.
RNF 02 - Escalabilidade: o sistema deve ser capaz de lidar com um aumento no número de usuários, dispositivos e operações sem comprometer o desempenho. Ele deve ser escalável para acomodar um crescimento futuro de demanda sem perda de eficiência

RNF 03 - Flexibilidade: O sistema deve ser flexível o suficiente para se adaptar a diferentes configurações de estacionamento e requisitos específicos de cada local.
RNF 04 - Resistência a falhas: O sistema deve ser robusto e capaz de lidar com falhas de componentes individuais sem impactar significativamente a funcionalidade global. Deve ser projetado para garantir a disponibilidade contínua, mesmo em situações de falha, com mecanismos de recuperação rápidos.
RNF 05 - Segurança: O sistema deve garantir a proteção dos dados do usuário.
RNF 06 - Portabilidade: o sistema precisa ser capaz de ter clientes desenvolvidos para várias plataformas (como web, desktop ou dispositivos móveis).
RNF 07 Manutenibilidade: o sistema deve ser de fácil de realizar atualizações.

2.2 Atributos de Qualidades Prioritários

Para garantir a ciência de contexto o usuário precisa fazer login e permitir acesso em tempo real da sua localização. Com isso em vista, o atributo de qualidade de maior prioridade para o projeto é a **segurança** dos dados fornecidos pelo usuário.

Ainda sob o ponto de vista da qualidade, é necessário incluir na arquitetura elementos que contribuam com a **usabilidade** do sistema, sendo o segundo atributo de maior prioridade no projeto, visando uma experiência de uso mais agradável.

Em terceiro nível de prioridade estão os atributos de **escalabilidade**, sendo capaz de lidar com um aumento no número de usuários, dispositivos e operações sem comprometer o desempenho. Em quarto nível de prioridade o de **portabilidade** garantindo a disponibilidade da aplicação em diferentes plataformas. E em quinto nível manutenibilidade para facilitar na manutenção o software durante todo ciclo de vida.

2.2.1 Métricas para avaliar os atributos de qualidade

RNF 01 - Usabilidade:

- Tempo de Aprendizado: avaliar o tempo que um usuário leva para se familiarizar e aprender a usar o sistema.
- Taxa de Erros: número de erros cometidos pelos usuários durante a operação do sistema, indicando a eficiência e facilidade de uso.
- Satisfação do Usuário: pesquisas ou avaliações para medir a satisfação geral dos usuários em relação à usabilidade do sistema.

RNF 02 - Escalabilidade:

- Desempenho sob Carga: tempo de resposta e eficiência quando há aumento significativo na carga de usuários.
- Capacidade de Usuários Concorrentes: número máximo de usuários que podem interagir simultaneamente com o sistema sem degradação significativa no desempenho.

RNF 05 - Segurança:

- Taxa de Incidentes de Segurança: número de incidentes de segurança relatados em um determinado período.
- Controle de Acesso: medir a eficácia dos mecanismos de controle de acesso para prevenir acessos não autorizados.

RNF 06 - Portabilidade:

- Facilidade de Atualização: Medir a facilidade com que o software pode ser atualizado para versões mais recentes em diferentes plataformas.
- Requisitos de Recursos: Avaliar os recursos (CPU, memória, etc.) necessários para o software funcionar em diferentes plataformas.

RNF 07- Manutenabilidade:

- Facilidade de Entendimento: obter índices de manutenabilidade que combinam várias métricas para fornecer uma pontuação geral.
- Tempo de Correção de Defeitos: Medir o tempo médio necessário para corrigir problemas ou defeitos no código.

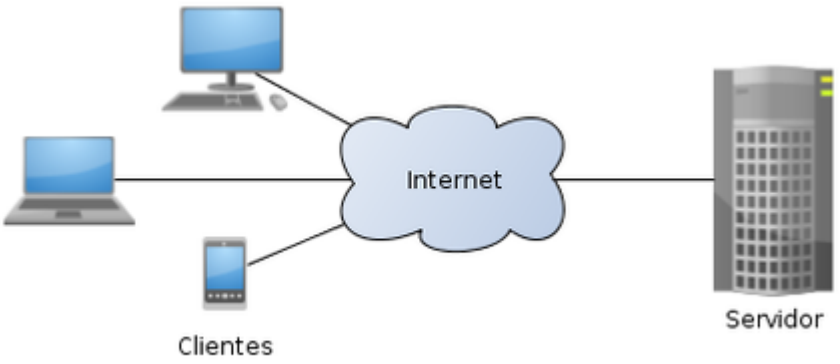
2.3 Tecnologias

Linguagem de programação back-end: Ruby
Linguagens para o front-end: HTML, CSS, Javascript e Dart
Ambiente de execução: Node.js
Frameworks: Bootstrap, Ruby on Rails e Flutter
Autenticação: Firebase
Nuvem: DigitalOcean
Hospedagem para controle de versão: Github
Sensores: Proximidade e GPS de celulares Android

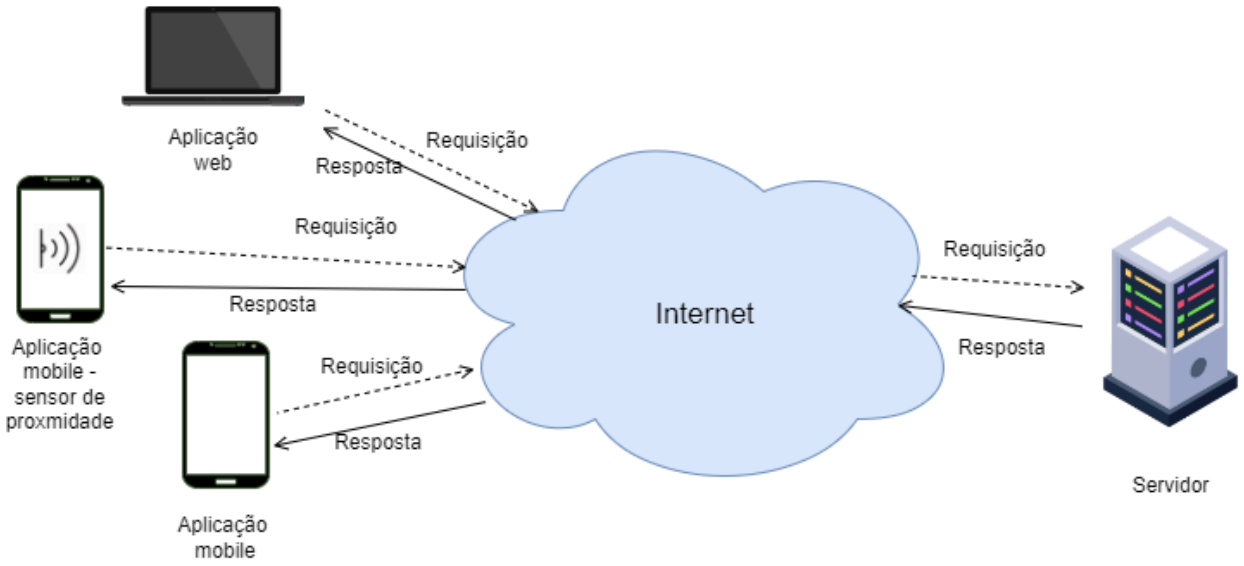
3. Representação da Arquitetura Candidata

O padrão escolhido para implementação do software SmartPark, foi o cliente-servidor. Esse é um modelo de design de software em que os sistemas são divididos em dois componentes principais: o cliente, que representa a interface do usuário e solicita serviços, e o servidor, que fornece esses serviços, processa as requisições e gerencia os recursos. A comunicação entre o cliente e o servidor geralmente ocorre por meio de solicitações e respostas. Essa abordagem facilita a escalabilidade, manutenção e flexibilidade dos sistemas, permitindo a separação de preocupações e distribuindo as responsabilidades entre as camadas do cliente e do servidor.

3.1 Diagrama cliente-servidor do sistema



4. Visão Geral



4.1 Descrição

Na visão geral é criada uma representação visual que fornece uma visão geral de um sistema ou processo. Ele mostra as principais entidades envolvidas e suas interações de forma simplificada.

Na visão geral da distribuição do sistema, segue uma arquitetura cliente-servidor, onde o frontend (página web e aplicativos) atua como cliente e o backend como servidor. O frontend, representa a interface do usuário, permitindo visualizar e interagir com as vagas de estacionamento. Por outro lado, o backend consiste em uma API localizada em um servidor, gerenciando as operações CRUD das vagas, processando solicitações do frontend, como reservas efetuadas pelo administrador, e mantendo o estado atual das vagas.

4.2 Componentes Principais:

4.2.1 Página Web:

Permite ao usuário uma visualização digital do estacionamento (Gêmeo Digital), incluindo o estado das vagas (ocupado ou disponível). Caso o usuário for administrador,

ele também poderá reservar vagas e visualizar informações colhidas acerca do estacionamento.

4.2.2 Aplicativo de Vagas:

Permite ao usuário uma visualização digital do estacionamento (Gêmeo Digital), incluindo o estado das vagas (ocupado ou disponível). Também analisará a localização do usuário, no contexto deste projeto, o estacionamento do INF e irá direcionar o usuário para a vaga disponível mais próxima da entrada.

4.2.3 Aplicativo Sensor:

Permite a conexão com o sensor de proximidade do celular. Caso esse for acionado, o status da vaga correspondente será atualizada.

5. Decisões arquiteturais

5.1 Requisitos Arquiteturalmente Significativos

ID	RAS	Descrição do Cenário
01	Segurança	O sistema deve garantir a proteção dos dados do usuário.
02	Usabilidade	O sistema deve ser fácil de usar e entender.
03	Escalabilidade	O sistema deve ser capaz de lidar com um aumento no número de usuários, dispositivos e operações sem comprometer o desempenho.
06	Portabilidade	O sistema deve ser compatível em diferentes plataformas

5.2 Decisões para o Front

ID do RAS	01
Tipo de decisão arquitetural	Usabilidade
Decisão	O sistema SmartPark usará Bootstrap para apoiar na usabilidade.
Justificativa	O uso do Bootstrap em um sistema é motivado por sua capacidade de criar interfaces web responsivas e visualmente atraentes. Trade-offs: + Rápido + Consistência - Customização
Forma de implementação (tecnologia)	Utilizando os componentes e estilos do Bootstrap para construir a interface do usuário. Aproveitando os elementos predefinidos, como botões, formulários e barras de navegação.

ID do RAS	02
Tipo de decisão arquitetural	Integração
Decisão	O sistema SmartPark deve se comunicar com um API, usando uma biblioteca que faça requisições HTTP
Justificativa	A capacidade de se comunicar facilmente com outras APIs é essencial para o funcionamento de listar as vagas a atualizá-las, com uso de uma biblioteca que faça requisições HTTP de maneira simples. Trade-offs: + Flexibilidade +Tratamento de Erros - Personalização Limitada

Forma de implementação (tecnologia)	Uma possível implementação seria com o uso da biblioteca axios, podendo fazer requisições HTTP de forma clara e eficaz.

ID do RAS	03
Tipo de decisão arquitetural	Existencial
Decisão	O sistema SmartPark deve utilizar o estilo arquitetural de componentes para apoiar na manutenibilidade do Frontend
Justificativa	Permite criar uma arquitetura organizada, onde cada componente representa uma funcionalidade ou elemento visual específico. Trade-offs: + Manutenibilidade + Modularidade + Reutilização de Código - Desempenho - Overhead de Gerenciamento
Forma de implementação (tecnologia)	Uma possível implementação seria com o uso do framework Vue.js, que permite construir interfaces complexas de maneira eficiente, promovendo a reutilização de código, a manutenibilidade e a separação clara de preocupações.

5.4 Decisões para o Backend

ID do RAS	04
Tipo de decisão arquitetural	Integração
Decisão	O sistema SmartPark deve se comunicar com uma API, usando uma biblioteca que faça requisições HTTP
Justificativa	A biblioteca Net::HTTP em Ruby destaca-se

	pela simplicidade e eficácia na manipulação de requisições HTTP. Integrada nativamente na linguagem, oferece uma interface direta e intuitiva para comunicação com APIs. Trade-offs: + Simplicidade + Eficácia - Concorrência
Forma de implementação (tecnologia)	Uso de tecnologia HTTP para comunicação de diversos sistemas através de requisições.

ID do RAS	05
Tipo de decisão arquitetural	Manutenibilidade
Decisão	O sistema SmartPark deve possuir código bem estruturado e documentado, de forma a facilitar futuras correções ou modificações no sistema
Justificativa	Uso do padrão de arquitetura MVC, que é um padrão que tem como um dos atributos de qualidade contribuir com a manutenção do software. Trade-offs: + Manter as atualizações isoladas do restante do sistema - Curva de aprendizado
Forma de implementação (tecnologia)	Uso do Framework Ruby on Rails que utiliza o padrão MVC em seus projetos que são desenvolvidos em Ruby.

6. Ponto de vista dos Casos de Uso

6.1 Descrição

Na análise de requisitos, é criada uma visão de casos de uso para fornecer uma base para o planejamento da arquitetura e outros artefatos. Essa visão representa os casos de uso e cenários que o usuário terá, além de classes e riscos técnicos relevantes para a arquitetura do sistema. A visão de casos de uso é considerada em cada iteração do ciclo de vida do software.

6.2 Visão de Casos de Uso

Os requisitos funcionais foram utilizados para compor os casos de uso, resumindo as principais funcionalidades do sistema nos diagramas abaixo:

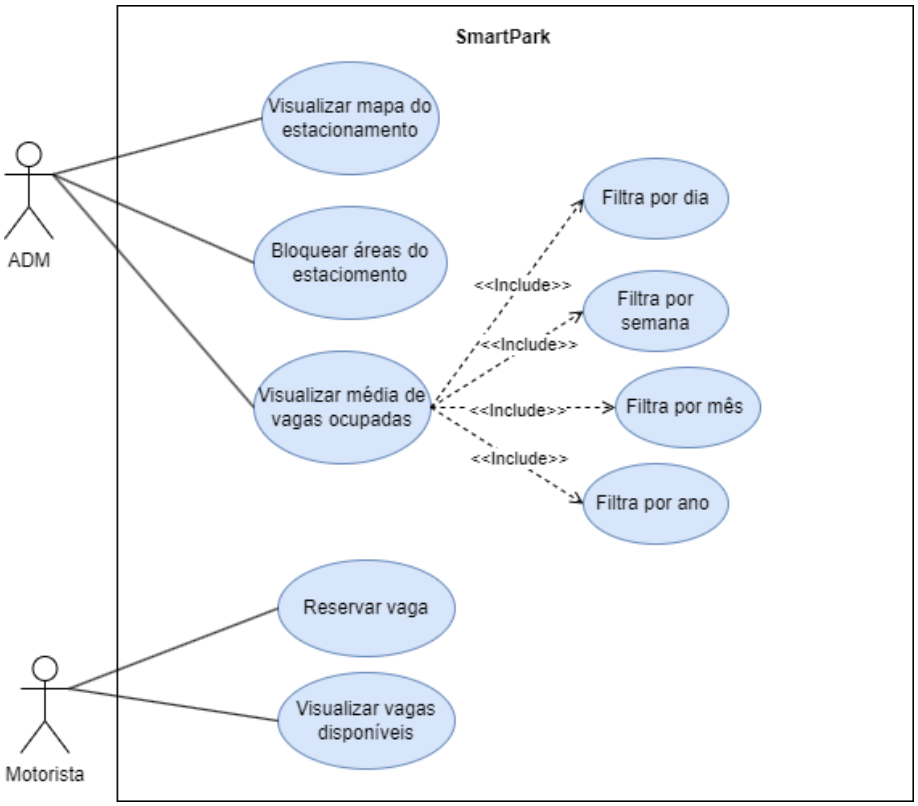


Figura 3: Casos de uso

7. Ponto de vista do Projetista

7.1 Descrição

O ponto de vista de projetista é um dos pontos de vista arquiteturais que se concentra nas decisões de projeto e nas preocupações técnicas específicas relacionadas à implementação do sistema. Ele fornece informações detalhadas e orientações para os projetistas e desenvolvedores envolvidos na construção do sistema.

7.2 Visão em Módulos

- **Módulo de Autenticação/Registro:**
  - Responsável pelo cadastro e autenticação dos usuários no sistema.
  - Funcionalidades: registro de novos usuários, login, recuperação de senha, gerenciamento de sessões.
- **Módulo das Vagas do estacionamento:**
  - Responsável por atualizar o status das vagas do estacionamento.
- **Módulo de Direcionamento a vaga disponível:**
  - Responsável por direcionar o usuário para a vaga de estacionamento disponível mais próximo

Estrutura Hierárquica:

1. Módulo de Autenticação/Registro
2. Módulo de Vagas do estacionamento
3. Módulo de Direcionamento à vaga disponível

8. Ponto de vista de Segurança

8.1 Descrição

É uma perspectiva específica dentro da arquitetura de um sistema que foca nos aspectos relacionados à proteção, privacidade e segurança das informações e dos recursos do sistema. Essa visão tem como objetivo identificar os requisitos de segurança, estabelecer as estratégias e controles de segurança adequados, e definir as medidas e práticas que devem ser implementadas para garantir a integridade, disponibilidade e confidencialidade do sistema.

8.2 Visão de Segurança

- **Autenticação e Controle de Acesso:**
  - Implementar um sistema de autenticação robusto para garantir que apenas usuários autenticados tenham acesso ao sistema.
  - Utilizar técnicas de controle de acesso para definir permissões e restrições de acordo com o papel ou perfil do usuário.

- **Proteção de Dados:**
  - Utilizar técnicas de criptografia para proteger dados confidenciais, como senhas de usuários e informações pessoais.
  - Garantir que as informações sensíveis sejam armazenadas de forma segura e acessíveis apenas para usuários autorizados.

- **Prevenção de Injeção de Código:**
  - Implementar mecanismos de validação e sanitização de dados para evitar ataques de injeção de código, como SQL injection e XSS (Cross-Site Scripting).

- **Proteção contra Ataques de Força Bruta:**
  - Implementar mecanismos de segurança que limitem o número de tentativas de login, como bloqueio de contas após várias tentativas falhas.

- **Proteção de APIs e Integrações:**
  - Utilizar autenticação e autorização adequadas para proteger as APIs utilizadas na integração com os serviços de streaming.
  - Validar e filtrar dados recebidos e enviados pelas APIs, evitando a exposição de informações sensíveis.

9. Ponto de vista do Fluxo de Dados

9.1 Frontend (Cliente):

- O usuário interage com a página web (frontend), visualizando o estado atual das vagas de estacionamento.

- Ao realizar uma ação, como fazer uma reserva, o frontend envia uma solicitação para o backend por meio de requisições HTTP.

9.2 Backend (Servidor/API):

- O backend, que consiste em uma API hospedada em um servidor, recebe as solicitações do frontend.
- Com base na solicitação, o backend realiza operações CRUD, como atualizar o status da vaga para reservada ou liberar uma vaga previamente reservada pelo administrador.
- Se houver integração com o sensor de proximidade do celular, o backend pode receber notificações do aplicativo móvel, alterando o status da vaga associada ao sensor.

9.3 Atualização do Frontend:

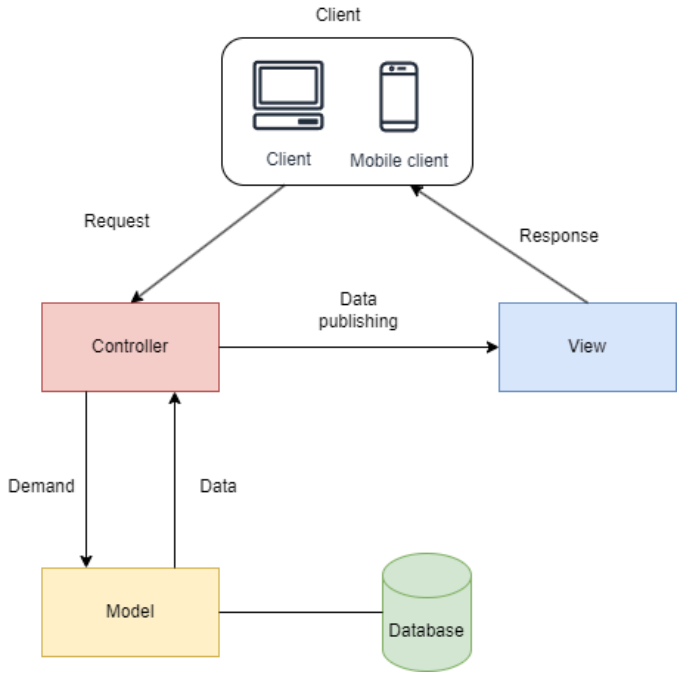
- Após processar a solicitação, o backend envia uma resposta ao frontend, fornecendo os dados atualizados ou confirmando a realização da operação solicitada.
- O frontend, por sua vez, atualiza dinamicamente a interface do usuário para refletir as alterações, como marcando uma vaga como reservada ou livre.

10. Ponto de vista Backend

10.1 Descrição

É a perspectiva do sistema responsável por gerenciar e processar os dados, lógica de negócios e operações que ocorrem no lado do servidor. É uma parte essencial, pois lida com a lógica de processamento e armazenamento de informações.

10.2 Diagrama



10.3 API

O backend é uma API desenvolvida usando o framework Ruby on Rails, muito utilizado para CRUD, utiliza o padrão MVC, (Model-View-Controller).

- Model (Modelo): gerencia dados e lógica de negócios.
- View (Visão): realiza a apresentação dos dados.
- Controller (Controlador): coordena interações, manipula entradas do usuário e atualiza modelo e a visão.
- Banco de dados: armazena dados da aplicação.

11. Aspectos de computação ubíqua contemplados

10.1 Ciência de contexto:

Utilizando a localização do usuário e comparando com a entrada do estacionamento, quando essa localização for a mesma iniciar a rota da vaga mais próxima.

10.2 Continuidade:

Com a adaptabilidade das informações de status da vaga, para *smartphones* uma visão em lista e para *desktop* um mapa.

10.3 Consistência:

Com uma UI e experiência do usuário que é consistente entre as versões *mobile* e *desktop.*

10.4 Complementariedade:

Com múltiplos dispositivos de interface que se complementam - versão *mobile* do administrador com uma lista das vagas do estacionamento e a versão *desktop* com um mapa do estacionamento e dashboard

10.5 Descrição do conceito de gêmeo digital será integrado na aplicação/sistema.

O gêmeo digital representará o estacionamento físico, dando uma visualização do estacionamento, das vagas disponíveis e seus status ao usuário cliente e admin. Ele também irá indicar o melhor caminho para a vaga disponível mais próxima. Além disso, o gêmeo digital irá colher os dados sobre seu gêmeo físico, como: tempo de permanência, ocupação durante o dia, meses com mais movimento, etc.