DOCUMENTO DE ARQUITETURA

LOCAL BONNUM

Versão 1.0.0

HISTÓRICO DE REVISÕES

Data	Versão	Descrição	Autor
17/01/2019	0.1	Criação do documento	Jesiel Viana

ÍNDICE

1. INTRODUÇÃO

O objetivo deste documento é fornecer um projeto de arquitetura detalhada da plataforma de *marketplace* de criptomoedas Local Bonnum, concentrando-se em quatro atributos-chave de qualidade: usabilidade, disponibilidade, segurança e testabilidade. Esses atributos foram escolhidos com base em sua importância no projeto e na construção do aplicativo.

2. VISÃO GERAL

Este documento abordará os requisitos funcionais significativos do ponto de vista da arquitetura. Cada um dos atributos de qualidade mencionados acima será descrito por meio de um conjunto abrangente de cenários seguido por uma visão geral da arquitetura, que inclui uma visão geral e uma descrição completa dos padrões e táticas que serão usados para abordar os principais atributos de qualidade. Isso será seguido por uma descrição de algumas visões do sistema. Finalmente, agradecimentos, referências e apêndices complementarão o documento.

A intenção deste documento é ajudar a equipe de desenvolvimento a determinar como o sistema será estruturado no mais alto nível. E também o gestor do projeto poderá usar este documento para validar que a equipe de desenvolvimento está atendendo aos requisitos acordados durante sua avaliação dos esforços da equipe.

3. REQUISITOS FUNCIONAIS

Muitos dos recursos envolvem salvar, atualizar ou visualizar dados de formulários, portanto, precisam ser considerados na arquitetura devido à quantidade de interações com o banco de dados. O sistema deve suportar gravação e leituras simultâneas no banco de dados.

O sistema também tem a necessidade de interagir com várias APIs externas. O sistema deve interagir com a rede Blockchain para manipular transações. Além disso, embora os serviços específicos sejam desconhecidos neste momento, é provável que o sistema tenha que interagir com outras aplicações externas para validações de segurança. Esses recursos são significativos do ponto de vista da arquitetura, já que o sistema deve ser projetado de maneira modular, de modo que os serviços externos possam ser trocados com facilidade para lidar com futuras atualizações.

Uma visão geral dos requisitos funcionais do sistema estão representados do diagrama de caso de uso abaixo (Figura 1).

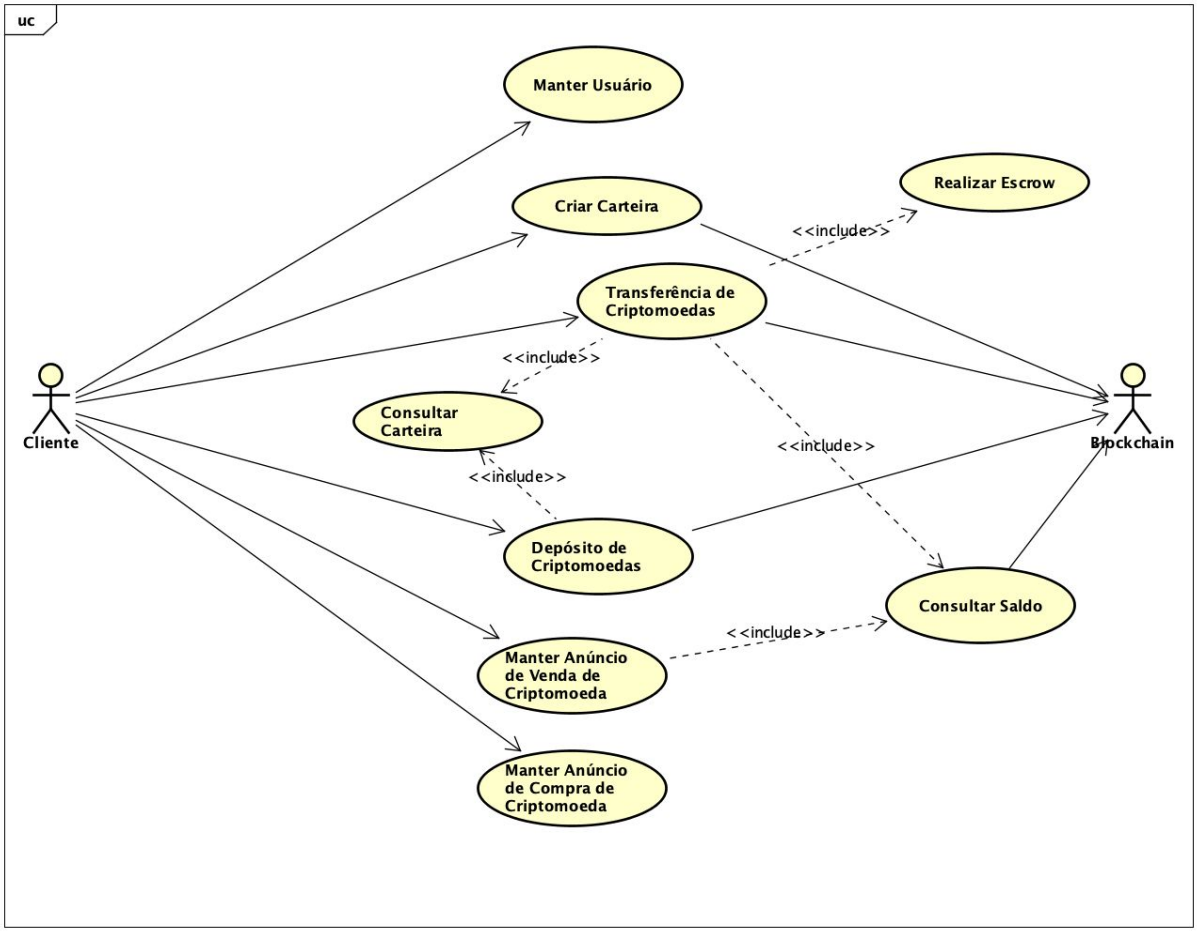

Figura 1 - Diagrama de Caso de Uso

Para uma descrição detalhada de todos os requisitos funcionais, consulte as Especificações de Histórias de Usuários.

4. REQUISITOS NÃO FUNCIONAIS

As tabelas a seguir descrevem cenários concretos para os quatro principais atributos de qualidade que devem ser incluídos no sistema.

4.1. USABILIDADE

A plataforma será construído para rodar em ambiente web com uma interface gráfica responsiva, que se adapta a diferentes dispositivos, em especial a diferentes resoluções de tela para uma melhor experiência do usuário. Abaixo são apresentados cenários concretos de usabilidade.

Cenário 01	O usuário final quer descobrir quais recursos estão disponíveis para eles.
Resposta	O sistema fornece uma interface familiar para o usuário e segue as boas práticas no design da interface do site para melhorar a capacidade de aprendizado.
Métrica	Número de erros na conclusão de uma tarefa, proporção de operações bem-sucedidas para operações totais

Cenário 02	O usuário deseja receber mensagens de erro do usuário e da situação apropriada quando ocorre um erro.
Resposta	O sistema fornece feedback visual informando se uma determinada ação foi bem-sucedida ou não. Se não obtiver êxito, o sistema fornecerá detalhes sobre o que deu errado e como corrigir a situação, quando possível.
Métrica	Satisfação do usuário, quantidade de tempo perdido, quantidade de dados perdidos.

Cenário 03	O usuário final deseja acesso rápido aos principais recursos da plataforma
Resposta	O sistema exibe informações pertinentes por padrão na tela inicial do usuário, sem forçá-las a acessar os menus aninhados para encontrar as informações que estão procurando.
Métrica	Tempo gasto para executar uma tarefa

Cenário 04	O usuário final deseja acessar a plataforma através de diferentes dispositivos
Resposta	O sistema exibe uma interface ajustada para o tamanho da tela disponibilizando uma interface padronizada e com os mesmos recursos em todos os dispositivos.
Métrica	Tempo gasto para executar uma tarefa em diferentes dispositivos, dispositivos mais utilizados para acesso à plataforma.

4.2. DISPONIBILIDADE

A disponibilidade é um importante requisito arquitetural para o sistema, pois o sistema deve estar disponível para que os clientes possam fazer transações de compra e venda de criptomoedas.. Se o sistema não estiver disponível para uso pode gerar falta de confiabilidade e perda de valores monetários para os clientes. Abaixo são apresentados cenários concretos de disponibilidade.

Cenário 01	Algum problema externo torna o servidor(hardware) da aplicação offline.
Resposta	A aplicação tem a capacidade de se inicializar junto com a inicialização do servidor(hardware).
Métrica	Porcentagem de tempo de atividade (mínimo de 95%)

Cenário 02	Um erro ou falha em algum componente da aplicação causa uma falha em todo o sistema.

Resposta	O sistema lida com todas e quaisquer exceções que possam ocorrer, para que o sistema possa falhar normalmente.
Métrica	Uma mensagem de erro significativa é registrada, indicando o que causou a falha do aplicativo.

4.3. Segurança

A plataforma deve implementar comportamentos básicos de segurança:

- Autenticação: usuários devem se autenticar usando um email e uma senha com no mínimo 8 caracteres;
- Autorização: somente usuários autenticados podem criar anúncios e realizar transações. Usuário só pode visualizar e movimentar dados da própria conta.
- Confidencialidade: os dados confidenciais devem ser criptografados (senha)
- Integridade de dados: os dados enviados pela rede não podem ser modificados (Uso de HTTPS).
- Auditoria: Nas versões posteriores, podemos implementar o registro de todas as ações dos usuários no sistema.

Abaixo são apresentados cenários concretos de segurança.

Cenário 01	Usuário não autenticado tenta realizar alguma transação.
Resposta	A aplicação solicita que o usuário faça o login como os dados cadastrados na plataforma.
Métrica	Usuários não autenticados conseguirem realizar alguma operação.

Cenário 02	Usuário autenticado no sistema tenta visualizar ou fazer transação com dados de outro usuário.
Resposta	A aplicação bloqueia a ação do usuário e registra log de auditoria.
Métrica	Quantidade de usuários autenticados no sistema que tentaram realizar alguma operação com dados de outra conta.

4.4. TESTABILIDADE

A testabilidade deve ser determinada em estágios iniciais de desenvolvimento. Devido à natureza iterativa de nossa metodologia, o teste de regressão será uma das principais formas de testes realizados. Portanto, o uso de teste unitários e testes de integração utilizando Stubs/Mocks é altamente valorizada para nós. Essa estratégia nos permitirá criar testes e usá-los rapidamente para testar o sistema quando alterações forem feitas ou novos recursos forem adicionados ao sistema.

Outra estratégia para testar nossos planos de equipe é separar o front end do back end. essa separação permite a substituição de implementações para vários

propósitos de teste. Isso também nos permitirá escrever testes para o back end sem ter que interagir com a interface gráfica. Abaixo é apresentado um cenário concreto de testabilidade.

Cenário 01	A cobertura de teste do sistema é inadequada.
Resposta	Forneça um ambiente de teste mais completo, que inclui testes automatizados e relatórios de cobertura.
Métrica	Porcentagem de linhas de códigos ou instruções executáveis cobertas por testes automatizados.

5. MECANISMOS ARQUITETURAIS

Um mecanismo arquitetural representa uma solução comum para um problema recorrente.

- Mecanismos de análise: representa a solução independente de tecnologia;
- Mecanismos de desenho: a solução com algum viés tecnológico;
- Mecanismo de implementação: solução concreta;

A tabela abaixo lista os mecanismos arquiteturais do projeto.

Análise	Design	Implementação
Estrutura da plataforma	Front end separado do back end	RESTful API com JSON
Front end	Interface responsiva de comunicação com o usuário da plataforma.	Bootstrap e React
Back end	Entidades do sistema, tratamento de rotas, classes de lógica e persistência de dados	Node.js com express e mongoose
Persistência	Banco de dados NoSQL	MongoDB
Integração com sistemas externos	Comunicação via serviços	RESTful API com JSON
Versionamento	Ferramentas para controle de versão	VS Code e Gitlab
Deploy	Integração contínua	Gitlab CI com Heroku

6. REPRESENTAÇÃO DA ARQUITETURA

As duas visualizações que detalhamos são: Visualização Lógica (Módulo) e Visualização de Processo (Componente e Conector). A Visão Lógica descreve a estrutura em camadas do sistema, enquanto a Visão de Processo descreve a estrutura do cliente-servidor do sistema.

A Visão Lógica mostra como o sistema é estruturado como um conjunto de unidades de código funcional, ou módulos, enquanto a Visão de Processo mostra como o sistema é estruturado como um conjunto de elementos computacionais que

possuem comportamento em tempo de execução (componentes) e interações (conectores).

6.1. VISÃO GERAL DA ARQUITETURA

A Figura 2 representa o diagrama de contexto do sistema, que mostra uma visão geral do sistema e como ele se encaixa no mundo em termos das pessoas que o utilizam e dos outros sistemas de software com os quais ele interage.

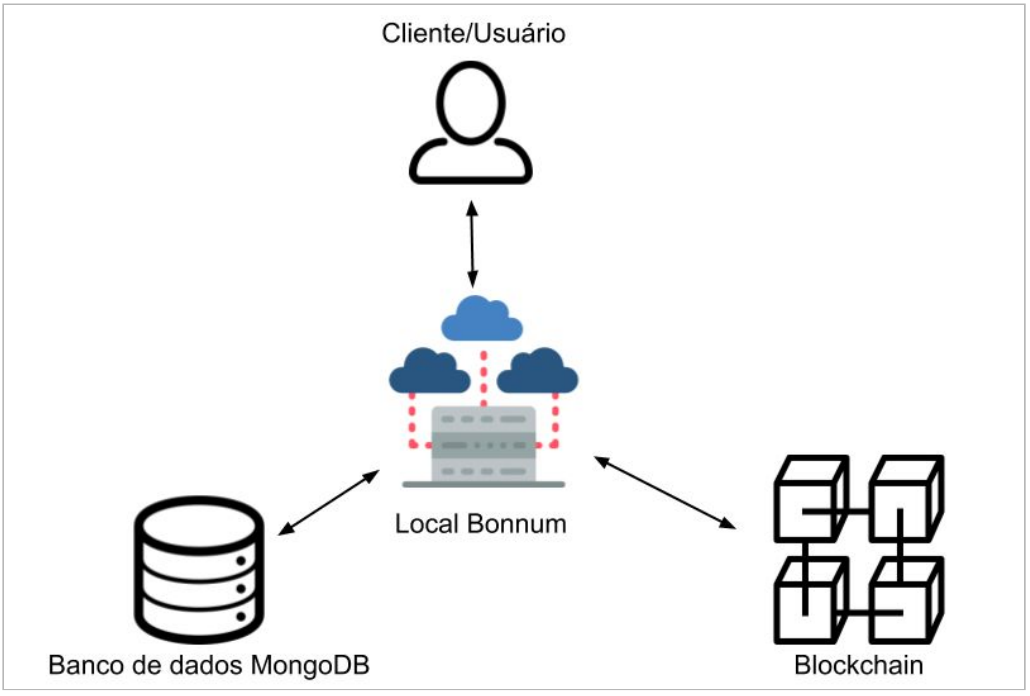

Figura 2 - Diagrama de contexto da plataforma Local Bonnum

Os principais componentes são descritos em detalhes de alto nível nas seções a seguir. Em suma, eles são divididos por camadas. Há a interface, a camada controladora, o modelo de domínio e a fonte de dados. Cada uma dessas divisões é vital para o sistema operar.

6.2. VISÃO LÓGICA

A Visão Lógica é uma visualização de nível superior do sistema que a Visão de Processos. A Visão Lógica, representada na Figura 3, mostra as camadas que compõem o sistema e a hierarquia dessas camadas. A camada de apresentação é encapsulada pelo cliente, enquanto as camadas inferiores são encapsuladas pelo servidor.

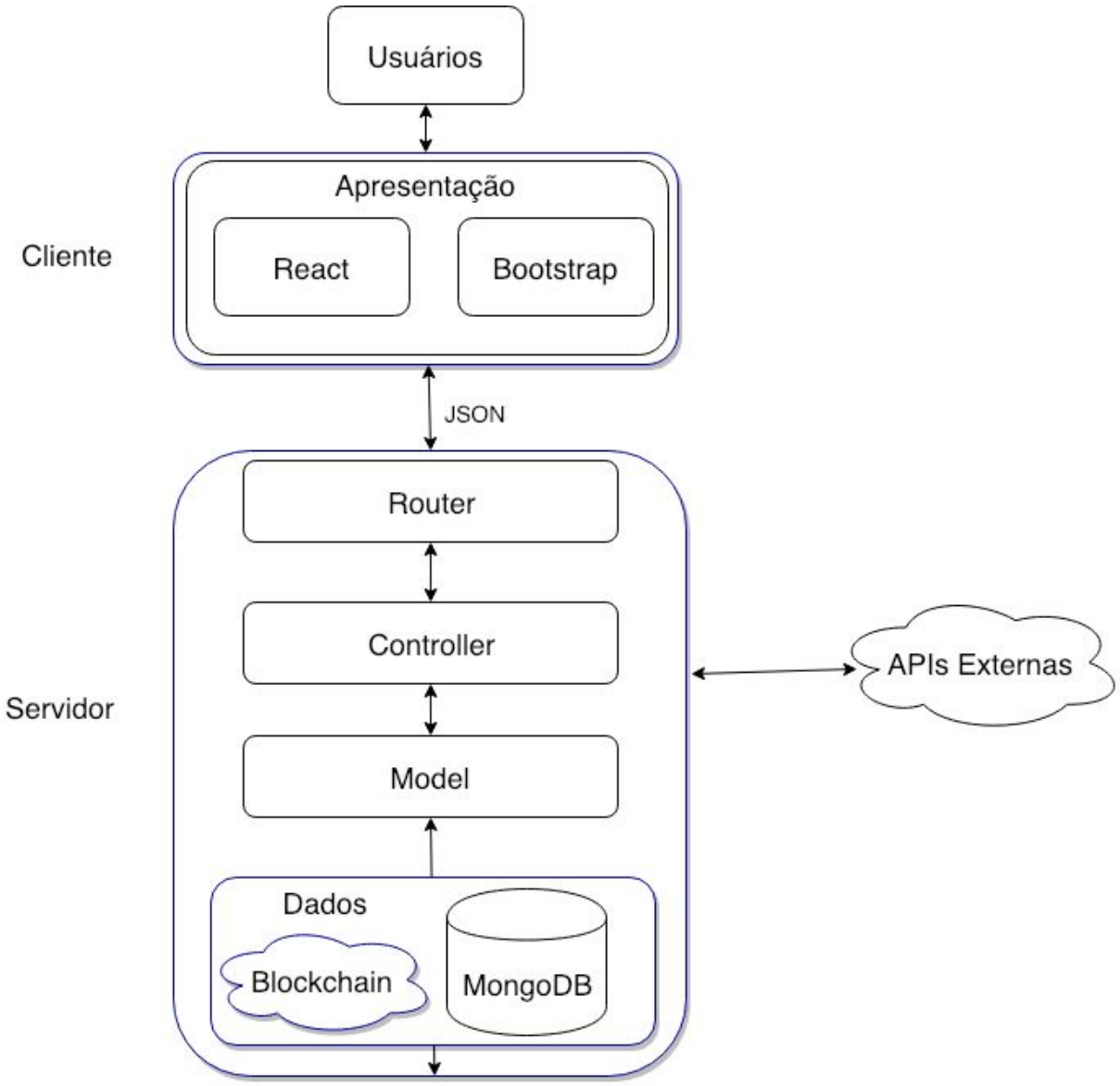

Figura 3 - Visão lógica do componentes e interações

Cada camada é denotada como retângulo. Os retângulos menores dentro representam um conjunto de elementos contidos em cada camada, porém não representam uma lista completa de entidades. Os grandes retângulos com bordas azuis demonstram a separação física das camadas entre cliente e servidor. Mais informações sobre essa divisão serão fornecidas na Seção 6.2. Abaixo será apresentado uma descrição de cada uma das camadas do sistema.

Camada de Apresentação

A camada de apresentação fornecerá ao usuário uma interface gráfica para interagir com o sistema. Ela será composta de HTML, CSS, JavaScript e outros arquivos relacionados que serão executados no navegador Web do usuário. O principal objetivo dessa camada é fornecer tudo o que o usuário precisa para

concluir suas tarefas no sistema. Os atributos de qualidade que dizem respeito à usabilidade e ao design de front-end terão maior impacto na apresentação.

Camada de Rotas

A camada de rotas é ponto de acesso para o cliente, ela é responsável por modularizar diferentes recursos da aplicação, implementada de acordo com o modelo REST. Cada rota atuará como fachada, servindo como uma interface geral para um recurso que pode ser acessado pela camada de apresentação. Essas fachadas (interfaces) fornecerão a capacidade de manipular ou corrigir o código sob cada funcionalidade, sem afetar a maneira como essa rota é chamada por outras camadas.

A modularização das rotas dentro do sistema também facilitará a identificação de possíveis defeitos, permitindo fácil manutenção. Cada rota pode ser testada individualmente, o que permite uma boa testabilidade em todo o sistema.

Camada de Controladores

A função da camada de controladores é implementar a lógica de negócios principal. Ela também servirá como conexão entre a camada de rotas e o modelo de domínio, mantendo assim a separação de interesses.

Certos recursos, como gerenciamento de carteira virtual e transferência de criptomoedas, exigem acesso a APIs externas. A camada de controladores será responsável por interagir com essas APIs externas.

Camada de Modelo de Domínio

O modelo de domínio contém todas as representações de objetos de dados do sistema no sistema, conhecidas como entidades. Isso também inclui métodos associados para qualquer objeto que contenha sua própria funcionalidade.

As entidades do modelo de domínio também são responsáveis pela coordenação de toda a comunicação entre os objetos na camada de domínio e suas respectivas coleções no banco de dados.

Camada de Dados

Todas as informações persistentes e qualquer dado oriundo de integração de API externa (por exemplo, Blockchain) formam a camada de origem de dados. Isso inclui o banco de dados MongoDB que junto com a Blockchain conterão todos os dados do sistema.

6.2.1. PADRÕES DE PROJETOS

Cliente-Servidor

Os clientes iniciam interações com servidores, que fornecem um conjunto de serviços. Os clientes chamam serviços conforme necessário a partir desses servidores e, em seguida, aguardam os resultados dessas solicitações. O cliente é

responsável por exibir e executar pequenas atualizações nos dados, enquanto o servidor gerencia o gerenciamento de dados.

O padrão cliente-servidor suporta a modificabilidade e a reutilização, uma vez que ele delineia serviços comuns, permitindo que eles sejam modificados de forma isolada. Com a centralização dos dados também facilita o controle de segurança. Os servidores podem controlar melhor o acesso a recursos, para garantir que apenas os clientes com credenciais válidas possam aceder e alterar os dados;

Model-View-Controller - MVC

O padrão Model-View-Controller (MVC) separa a interface do usuário das funcionalidades da aplicação. Com o MVC, a aplicação é dividida em três tipos de componentes: modelos, que contêm os dados da aplicação; views, que exibem os dados subjacentes e interagem com o usuário; e controladores, que atuam entre o modelo e a visão e gerenciam mudanças de estado.

O padrão MVC oferece suporte à usabilidade, pois permite que a interface do usuário seja projetada e implementada separadamente do restante da aplicação.

6.3. VISÃO DE PROCESSO

A Visão de Processo, representada na Figura 4, detalha as interações entre o cliente e servidor e os componentes específicos que compõem cada um deles.

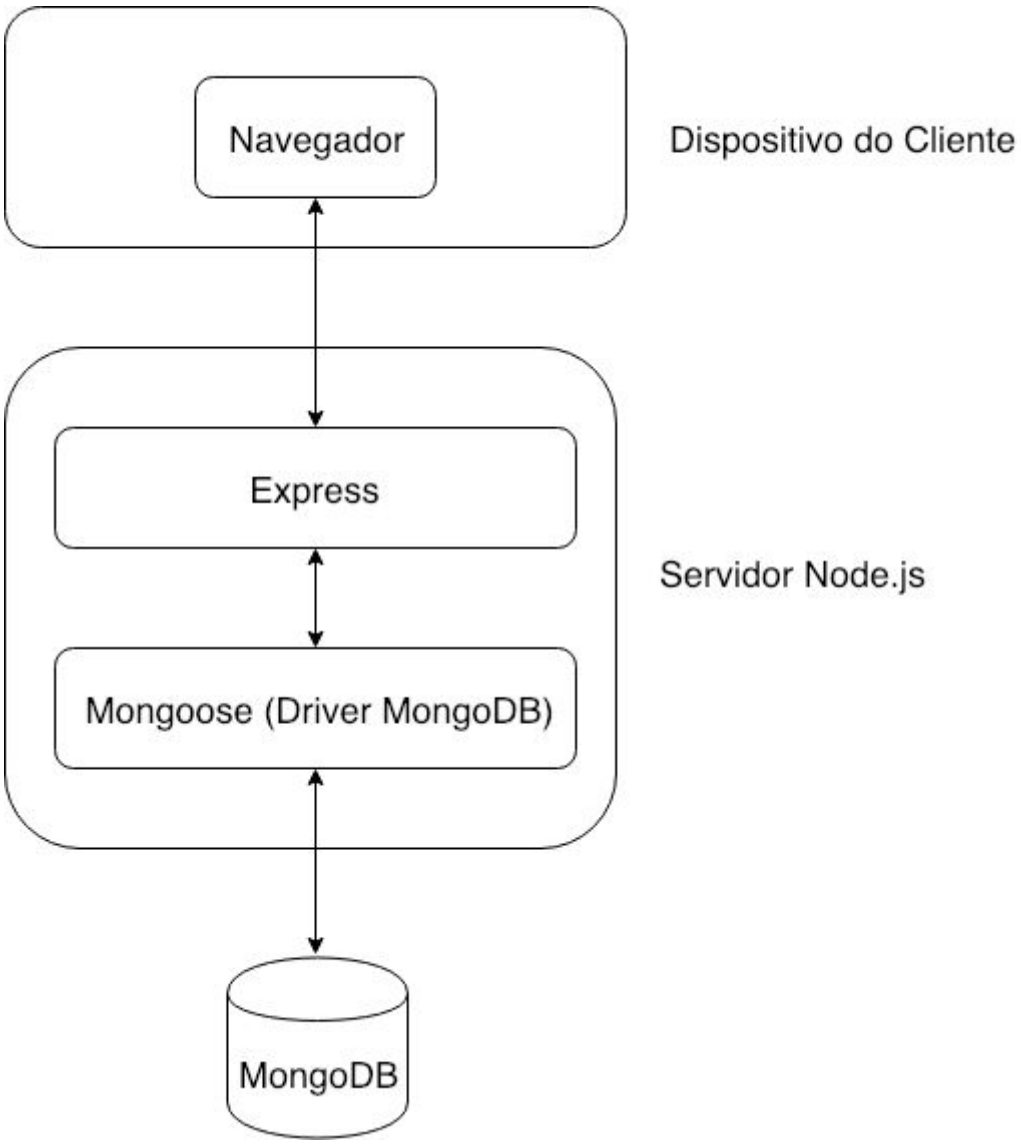

Figura 4 - Visão Cliente-Servidor

Abaixo será apresentado uma descrição de cada um dos elementos da Figura 4.

Cliente

Cliente é um componente que chama uma rota de um componente do servidor. No contexto da plataforma Local Bonnum, o cliente é um navegador da Web usado para acessar o sistema. O cliente faz solicitações HTTP usando serviços da web RESTful.

Servidor

Servidor é um componente que fornece serviços aos clientes. Os servidores possuem APIs que descrevem os serviços que eles fornecem. O Local Bonnum será implementado em um servidor Node.js. O servidor fornece serviços da Web por meio de HTTP, um protocolo de camada de aplicativo TCP / IP.

Comunicação entre Cliente e Servidor

Cliente e servidor se comunicam entre si usando um padrão de troca de mensagens request-response. O cliente envia uma solicitação ao servidor e o servidor envia uma resposta em retorno. Cliente e servidor usam uma linguagem comum de serviços da Web RESTful, para que o cliente e servidor saibam o que esperar. JSON é a linguagem utilizada na plataforma Local Bonnum para troca de mensagens entre cliente e servidor. Para lidar com várias solicitações de uma só vez, o servidor usa um sistema de agendamento para priorizar solicitações de entrada de clientes. O servidor também limita como um cliente pode usar os recursos do servidor para evitar um ataque de negação de serviço.