Trabalho de Arquitetura

“MyDebt”

Equipe: Jonas Deyvid - 385432, Jonas Lopes - 385433 e Maike Bezerra - 392070

Professor: Carlos Diego

UFC - Quixadá 2018

SUMÁRIO

VISÃO GERAL

Este documento é o documento de arquitetura do sistema MyDebt. Aqui serão mostradas informações de apoio aos stakeholders sobre o sistema em questão. As visões modulares, componente conectores e alocação de recursos. Esse documento não mostrará especificação de requisitos ou de casos de uso. Este é o principal artefato de consulta das partes interessadas acerca do sistema.

Também aqui neste documento você terá acesso linkado aos materiais de trabalho, como diagramas, visões modulares em alta resolução e outras imagens pela seção diretório.

MYDEBT

O sistema MyDebt surge para tornar mais fácil o gerenciamento de contas do dia a dia, reduzindo seu tempo lado a lado com os débitos, que te faz ter mais

tempo para fazer o que gosta e com um bônus: sem a preocupação de ficar decorando datas de vencimento e, por estresse, acabar esquecendo alguma!

O sistema conta com funcionalidades de cadastro de contas, lembrete de conta a vencer por email, histórico de contas pagas, cadastro de recebedores de contas, geração de boletos, visualização de boletos para fácil impressão, entre outras funcionalidades que facilitarão sua vida!

O sistema ao avisar sobre uma conta, também dará a opção para o usuário copiar o código de barras cadastrado com apenas um clique. Após isso ele poderá selecionar o internet banking que preferir e será redirecionado com facilidade para realizar o pagamento de sua conta.

O usuário após cadastrar sua conta, também poderá cadastrar os melhores dias para ser lembrado. Por padrão o MyDebt avisará exatamente no dia anterior e no dia do vencimento da conta. Tudo isso de maneira sutil para não estressar o usuário, que já tem várias outras contas para pagar. Sabemos que contas a pagar não é um dos assuntos mais satisfatórios e respeitamos isso.

OBJETIVOS DE NEGÓCIO

Tornar fácil o gerenciamento de contas:
O MyDebt tornará simples e fácil ao usuário, gerenciar suas pendências financeiras através de uma plataforma rápida e intuitiva.

Proporcionar ao usuário um ambiente simples que o lembre de seus compromissos sem estresse:
Através de notificações sutis por email, o sistema lembrará ao usuário na data estabelecida sobre o pagamento de sua conta.

ATRIBUTOS DE QUALIDADE

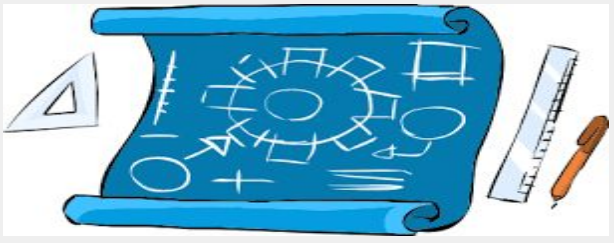
Usabilidade: O cadastro de uma conta no sistema deve ser feito em no máximo duas telas; A interface também precisa ser intuitiva. Vemos que isso é um sistema para cobranças sutis, tudo que o usuário menos quer é um estresse a mais, causado por uma interface feia e sem sentido.

Disponibilidade: O sistema deve estar disponível durante o período comercial sem interrupções. Isso se explica pois, se o sistema não estiver disponível em um nível satisfatório, ele pode deixar de relembrar o usuário, ferindo então um dos objetivos organizacionais.

Interoperabilidade: O MyDebt se comunicará com os sistemas de e-mails para enviar lembretes sutis aos usuários quando sua conta estiver por vir. Para isso será necessário esse atributo de qualidade interna para realizar essa comunicação e transação de mensagens.

Segurança: O MyDebt permitirá ao usuário realizar uma autenticação no sistema com um login e uma senha permitindo o mínimo de segurança em sua navegação pelo sistema e utilizando a tecnologia Spring Security.

REQUISITOS E RESTRIÇÕES ARQUITETURAIS



Padrões:

Utilizaremos como principal padrão arquitetural o padrão MVC - Model View Controller. A escolha deste padrão se dá pelo fato de que nosso sistema, solicitado pelo nosso cliente fictício, será um sistema web com funções de notificação através de mensagens por email. Esse sistema poderá evoluir para uma abordagem mobile com notificações pelo próprio celular quando uma conta estiver por vencer, com a construção de uma nova view. A facilidade para a implementação de uma nova view sem grandes impactos no sistema total, é a principal razão da escolha do padrão MVC.

Sabemos que o padrão MVC é um padrão que herda do padrão camadas. Neste caso, na seção de visões modulares poderá ser visto que foi adicionada uma outra camada, chamada persistence. Ela será melhor explicada nesta seção.

Também mesclaremos o sistema com o padrão cliente servidor, visto que será necessário o usuário consultar informações de contas cadastradas, boletos e recebedores em um banco de dados. O banco de dados rodará em um servidor, então a explicação para o uso de cliente-servidor se dá nesse aspecto.

O sistema também permitirá ao usuário filtrar as contas por valor, data de vencimento e etc.. Neste sentido, discutimos sobre o uso do padrão duto-filtro, porém, achamos inviável visto que uma única pessoa não tem tantas contas que justifique o uso de filtros múltiplos. Neste caso, haverá filtros individuais. Então, ou o usuário filtra por data, ou filtra por vencimento por exemplo.

Foi conversado também sobre o uso de micro serviços, e rapidamente concluímos que não é uma opção viável neste momento. Principalmente porque o sistema não é grande o suficiente que justifique ele ser modularizado em serviços menores funcionais.

Requisitos arquiteturais:

Requisito	Solução

Linguagem	Usaremos o framework Spring com a tecnologia Spring Boot para o desenvolvimento backend do código. Utilizaremos ele integrado com a linguagem Java. Para frontend usaremos Html, css, e Javascript. Algumas restrições serão executadas na própria interface, impedindo que o usuário entre em uma exceção. O css será reforçado com o Bootstrap. Utilizaremos Thymeleaf para o transporte dos objetos do frontend para o backend.
Plataforma	O sistema funcionará com base nos dois navegadores mais usuais Chrome e Mozilla com precisão. Não nos responsabilizamos por perdas em desempenho, disponibilidade ou usabilidade através da execução do sistema em outros navegadores. O sistema será responsivo, permitindo o usuário acessá-lo de um dispositivo móvel sem perdas consideráveis em usabilidade. Utilizaremos servidor Apache Tomcat 9.0 para subir a aplicação.
Segurança	Utilizaremos do Spring Security para desenvolvermos a camada de segurança. Utilizaremos mecanismo de autenticação de usuários do Spring Security. A criptografia das senhas de usuários também será gerenciada pelo Spring Security.
Persistência	A persistência dos dados será administrada pela biblioteca JPA com o framework Hibernate O banco de dados utilizado será o PostgreSQL.

Restrições arquiteturais:

Os e-mails relacionados ao recebimento dos boletos serão gerenciados pelo servidor de e-mail do usuário. Caso o e-mail não seja despachado para a caixa de entrada, será responsabilidade do usuário alterar isto, escolhendo a opção de falso spam.

O banco de dados inicialmente rodará em um servidor da aplicação (que será um computador usado como servidor.) Futuramente este servidor

será mais robusto e na nuvem, como por exemplo um Azure ou AWS da Amazon. Independente do servidor escolhido a instância da base será a mesma, o PostgreSQL. Essas decisões serão tomadas antes da publicação oficial do sistema.

STAKEHOLDERS

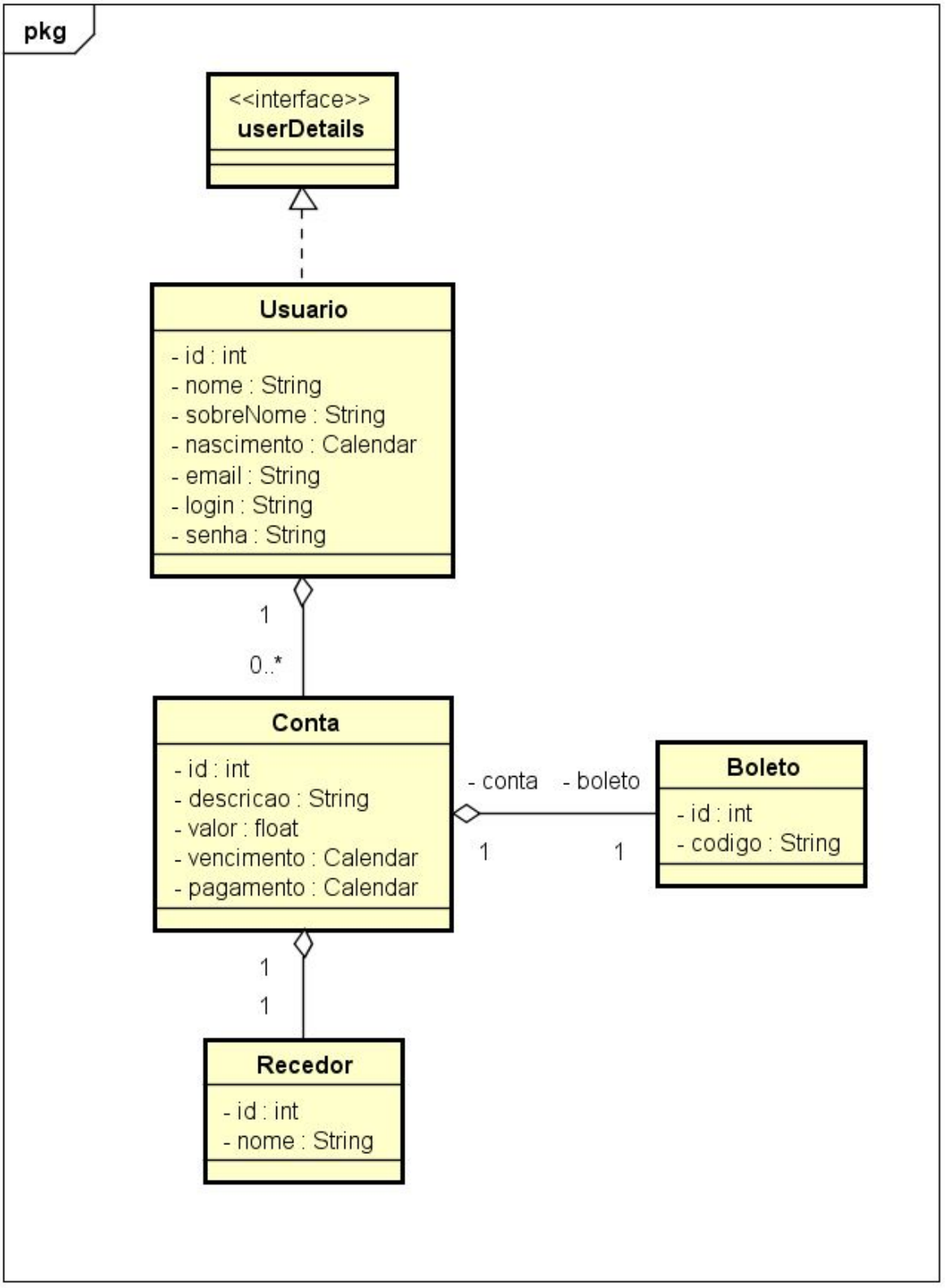
Desenvolvedores: Time de desenvolvimento, neste caso a equipe do trabalho que será responsável por discutir e priorizar os requisitos funcionais e não funcionais. Também os atributos de qualidade definidos pela arquitetura e que aproximem o sistema dos objetivos de negócio do sistema.

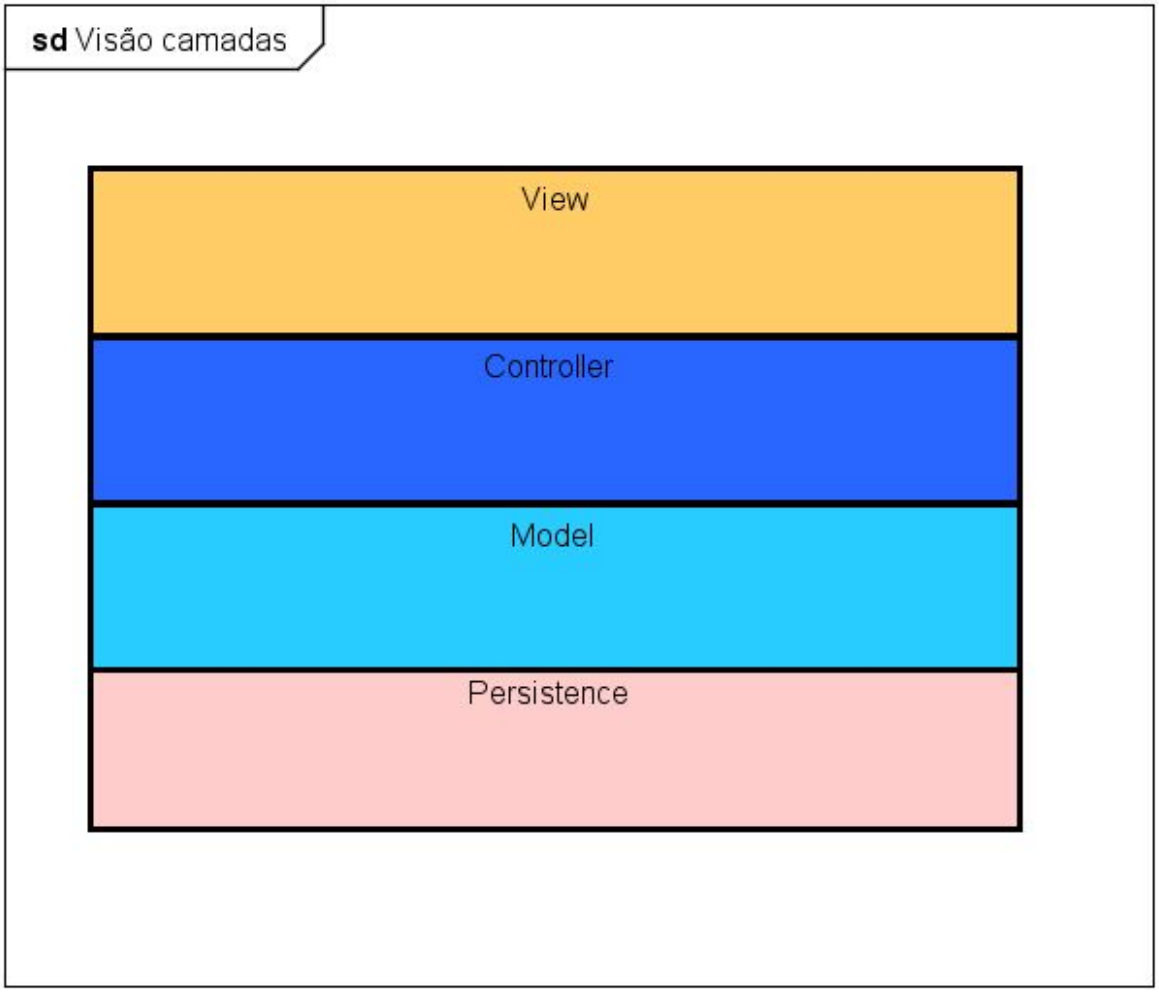
Usuário final: Nosso usuário final e público alvo são as pessoas que precisam se organizar financeiramente ou que queiram simplesmente organizar e armazenar suas contas em um sistema que as livre da responsabilidade de se lembrar de pagar cada conta, já que o sistema fará isso.

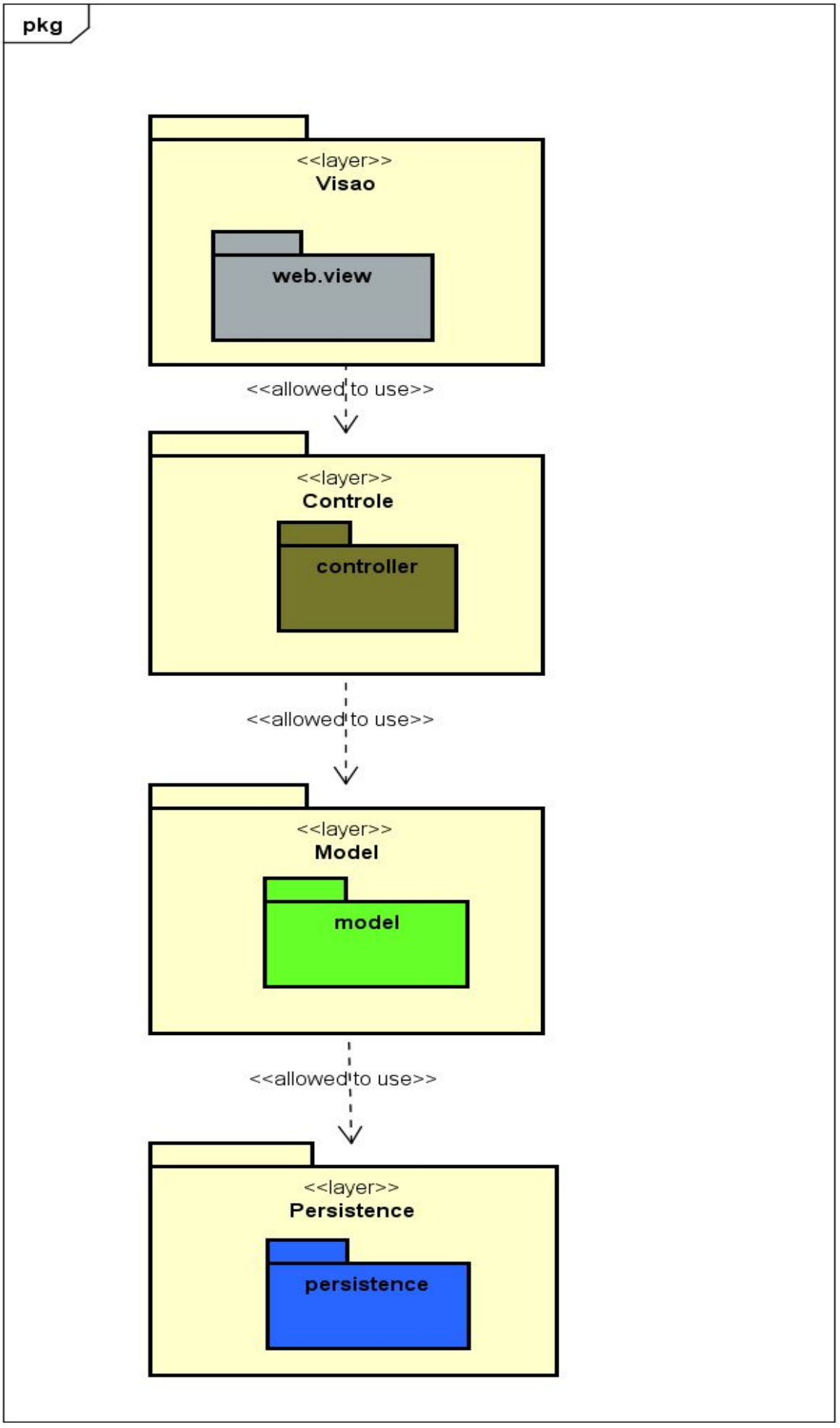
Desenvolvedores open-source

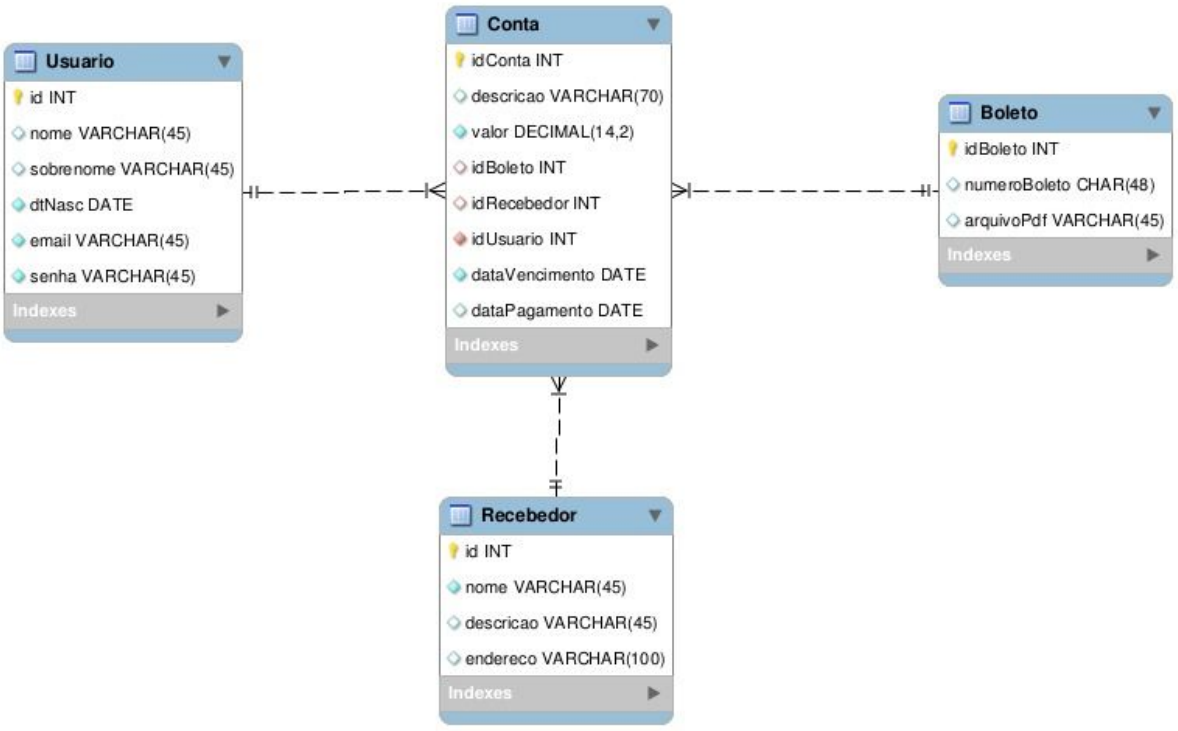
A arquitetura deve estar simples e entendível para que outros desenvolvedores possam reutilizar o código na criação dos seus próprios sistemas variantes. Contribuindo assim para a comunidade desenvolvedora.

VISÕES MODULARES


pkg
<<interface>>
userDetails
Usuario
- id : int
- nome : String
- sobreNome : String
- nascimento : Calendar
- email : String
- login : String
- senha : String
1
0..*
Conta
- id : int
- descricao : String
- valor : float
- vencimento : Calendar
- pagamento : Calendar
- conta
- boleto
1
1
Boleto
- id : int
- codigo : String
1
1
Recedor
- id : int
- nome : String

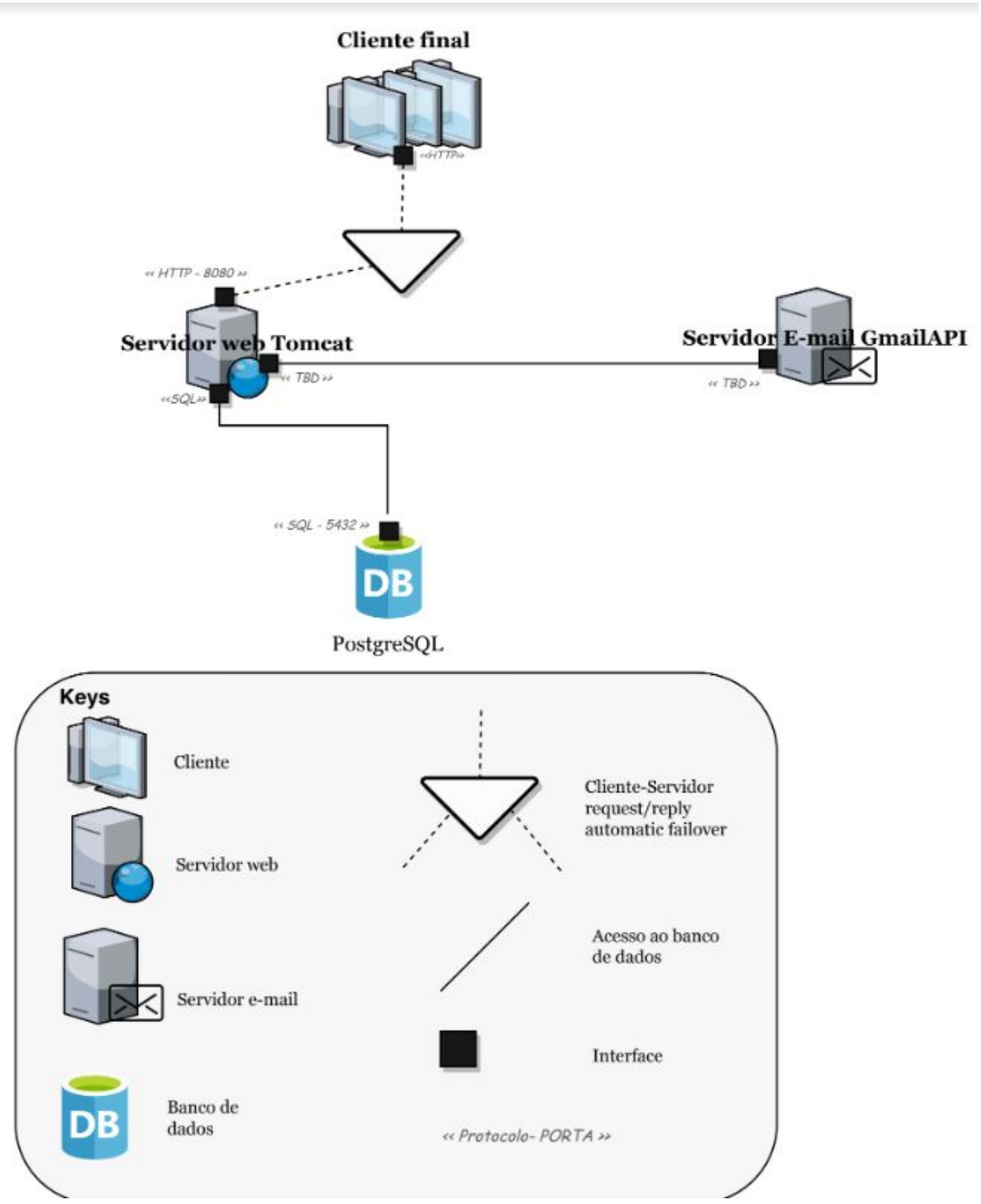

sd Visão camadas
View
Controller
Model
Persistence


pkg
<<layer>>
Visao
web.view
<<allowed to use>>
<<layer>>
Controle
controller
<<allowed to use>>
<<layer>>
Model
model
<<allowed to use>>
<<layer>>
Persistence
persistence

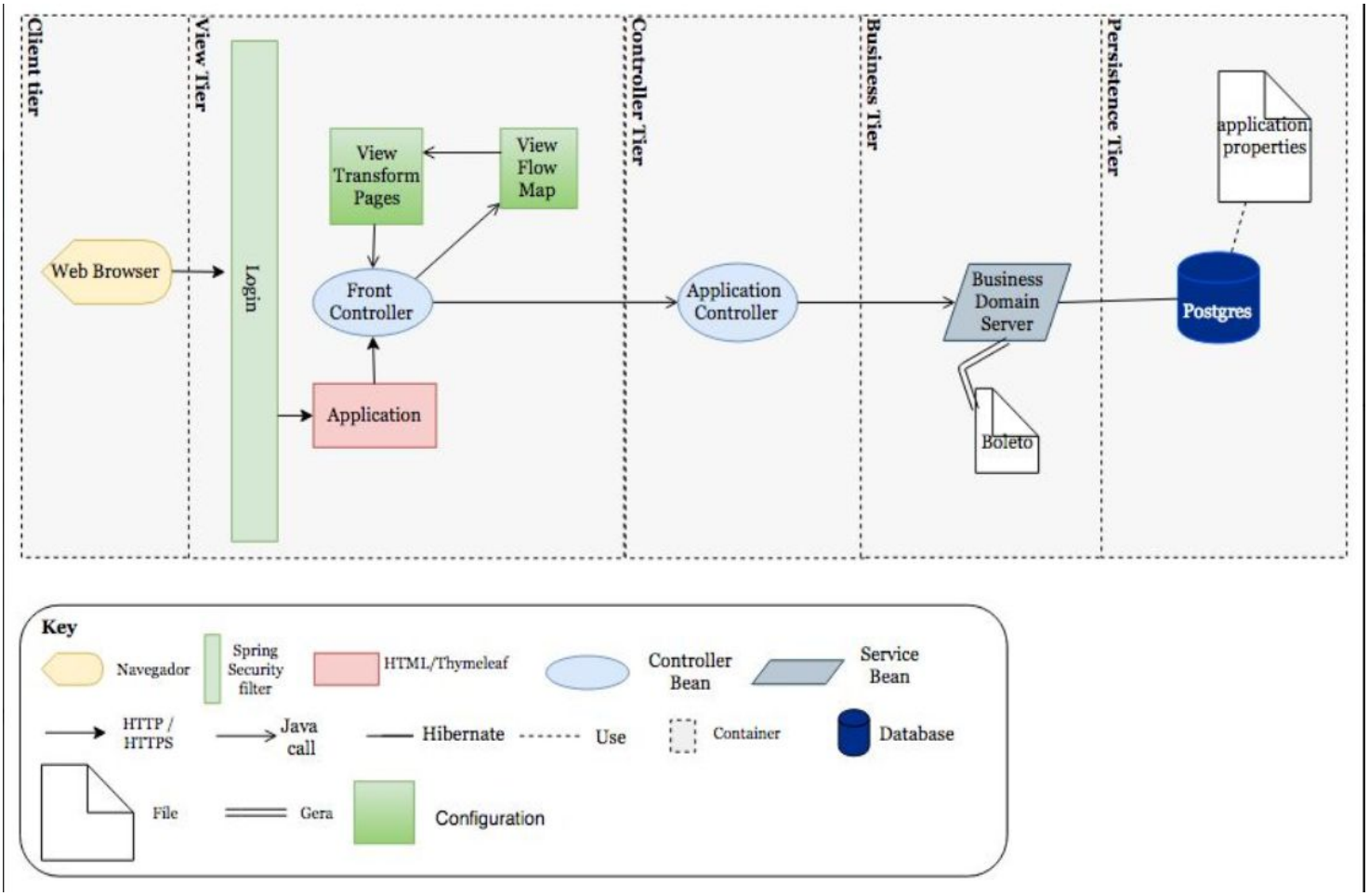


VISÕES C & C

Visão geral Componente e Conector sistema MyDebt.



Visão final Componente e Conector sistema MyDebt.



Utilizamos o Spring Boot e então alguns dos componentes presentes na **Key** são gerenciados pelo próprio Spring, como por exemplo o Login, View Transform Pages, View Flow Map e Front Controller.

Diagramas de sequência para os fluxos principais do sistema.
Como o diagrama tem uma largura maior que a visualização do Google Docs, disponibilizamos os links para acesso aberto em uma melhor resolução.

Fluxo de cadastro de conta - Clique aqui
Imprimir boleto - Clique aqui
Notificação de conta - Clique aqui

Caso queira o arquivo completo para abrir com o Astah Uml - Clique aqui

Porque não usamos publisher subscribe?

“Para que serve o publisher subscribe?

O estilo publisher subscriber é usado para enviar eventos e mensagens para um conjunto desconhecido de destinatários. Como o conjunto de destinatários do evento é desconhecido para o produtor do evento, a exatidão do produtor não pode depender desses destinatários. Assim, novos destinatários podem ser adicionados sem modificação aos produtores.”

VISÕES Alocação de Recursos

Como já explicado no índice, essa visão é mais dedicada a stakeholders técnicos como por exemplo, desenvolvedores e designers. Nesta visão mostramos a parte de alocação de hardware direcionada ao sistema em questão, MyDebt.

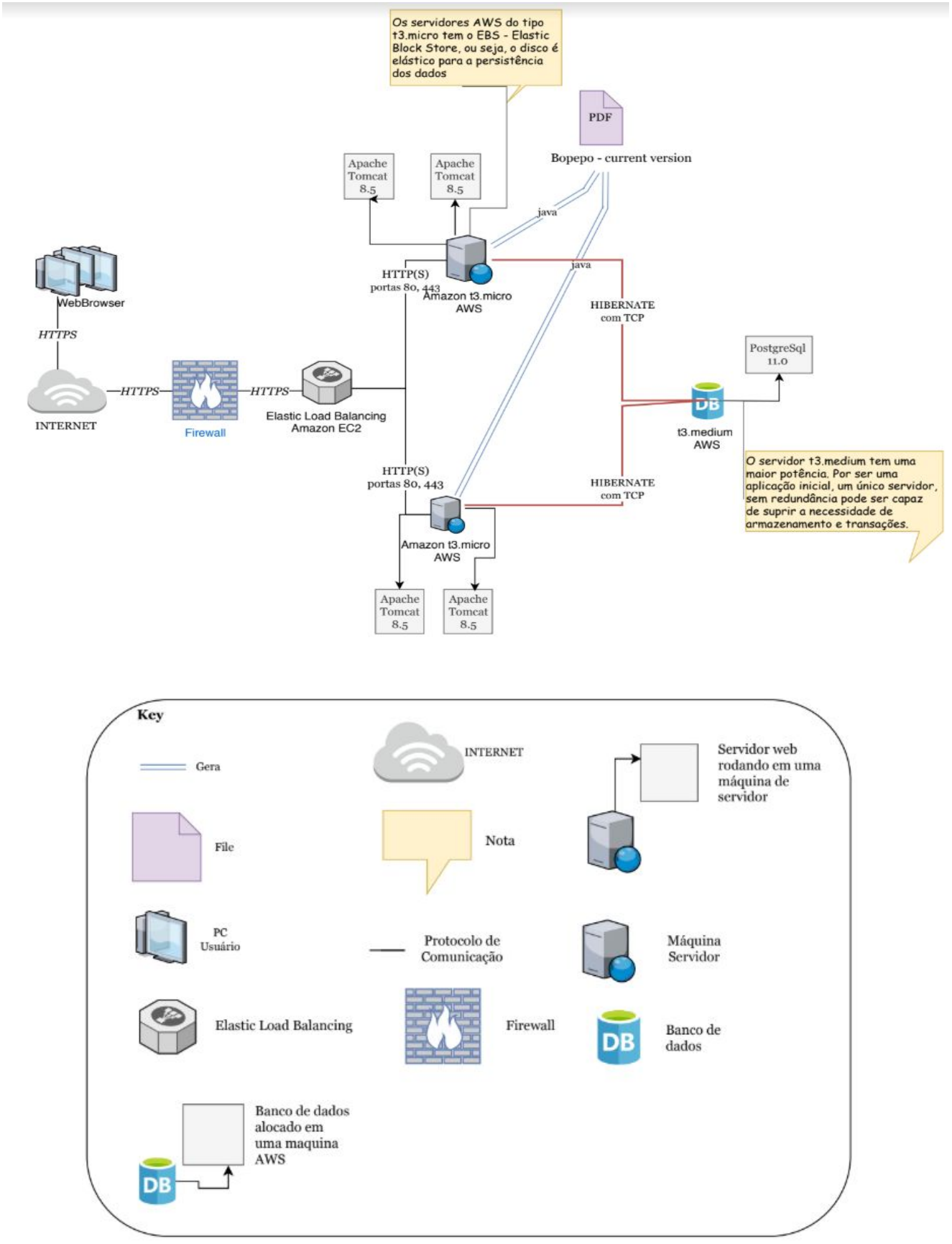


Diagrama de implantação.

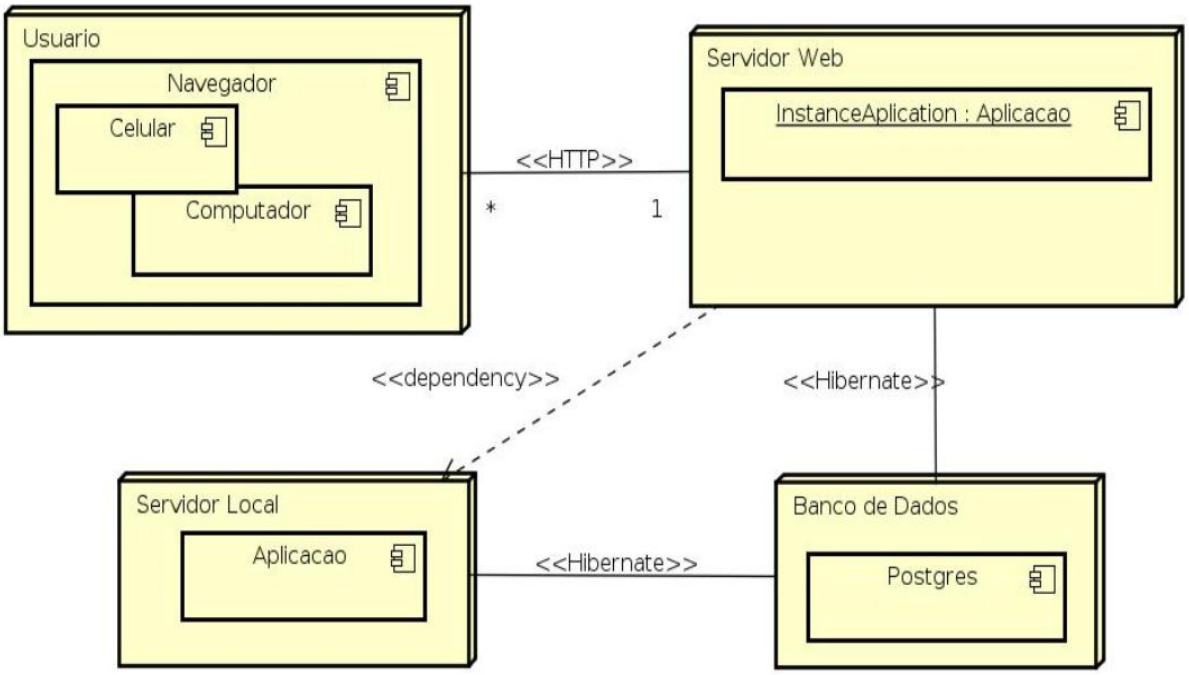


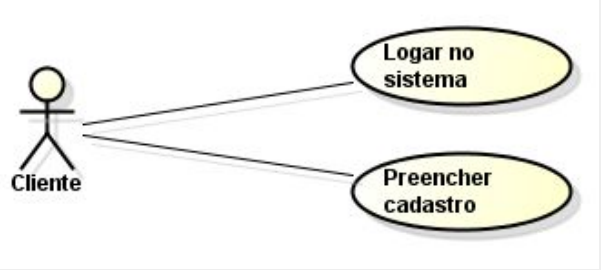
Tabela de alocação de recursos de hardware para a aplicação.

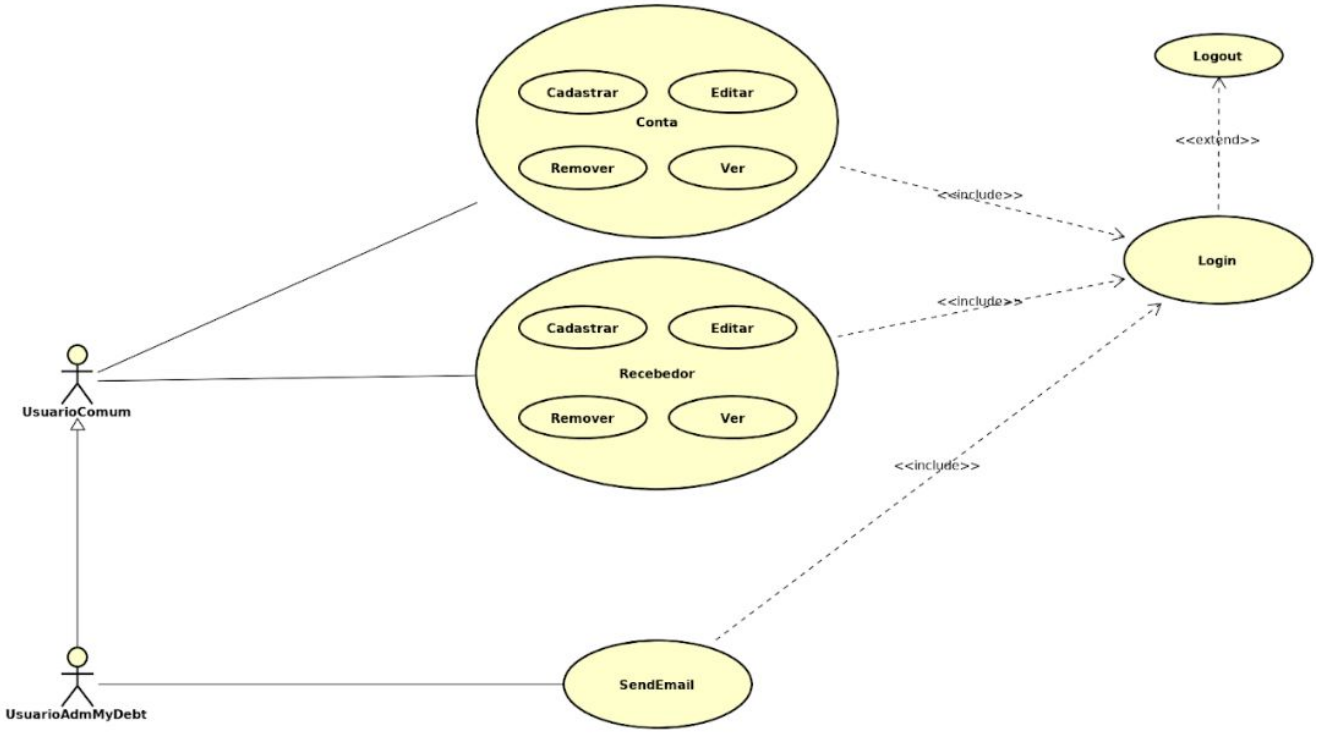
Atributos de hardware	Escolha para cada hardware
Banda da internet	1 mbps de download, ao menos 100Kbps upload
Disco rígido	Não é necessário se preocupar pois é um sistema online. O AWS elastisa o armazenamento.
Processador	Pelo menos dois núcleos com frequência de 1,5 GHz
Memória	800 MB Ram independente da frequência. Tipo: DDR2 acima.
Servidor versão	Apache Tomcat 8.5
Restrições	O usuário deve realizar qualquer ação no sistema, sem levar mais de 5 etapas.
Segurança crítica	Um usuário só poderá acessar a página inicial caso crie uma conta ou já tenha uma conta e se logue nela.
Gatilho de migração	Se a página inicial demorar mais de 20 segundos para ser carregada, deve-se abrir uma nova

	instância do Apache Tomcat. (Acreditamos que essa carga acontecerá com pouquíssima frequência.)
Servidor para aplicação web	Amazon AWS t3.micro vCPU: 2 Créditos de CPU/hora: 12 Memória: 1 GB Armazenamento: EBS - Elastic Block Store Performance de rede: Até 5 Gbps
Servidor para aplicação web	Amazon AWS t3.medium vCPU: 2 Créditos de CPU/hora: 24 Memória: 4 GB Armazenamento: EBS - Elastic Block Store Performance de rede: Até 5 Gbps
Banco de dados	PostgreSql 11.0
Geração de boletos	Bopepo current version

*Créditos de CPU/hora no caso dos servidores AWS é uma tecnologia que armazena os períodos ociosos do processador e transforma isso em horas de uso de processamento. Então, se o processador do servidor passar algumas horas ociosos, essas horas poderão ser usadas em processamentos de pico como se fosse uma troca entre o período ocioso e o período de sobrecarga. Com isso, o sistema terá o processamento que necessita, mesmo que esteja em momentos de stress no processador. Tudo isso claro, com um limite que é dado pela medida Créditos de CPU/hora.

Casos de Uso





DIRETÓRIO

Aqui se localizam os arquivos fontes usados nas visões modulares. Clique nos links para ser redirecionado aos arquivos.

Diagrama Relacional: Clique aqui...
Diagrama de Classe: Clique aqui...
Diagrama geral componente conector alta resolução: Clique aqui...
Diagrama final componente conector alta resolução: Clique aqui...
Apresentação em slide dos diagramas C&C: Clique aqui...