



# Universidade Norte do Paraná

SISTEMA DE ENSINO PRESENCIAL CONECTADO  
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

MARCELO EDUARDO ALVES PAIXÃO RESENDE

## DESENVOLVIMENTO DE SISTEMAS II

MARCELO EDUARDO ALVES PAIXÃO RESENDE

## **DESENVOLVIMENTO DE SISTEMAS II**

Trabalho em grupo apresentado à Universidade Norte do Paraná - UNOPAR, como requisito parcial para a obtenção de média semestral nas seguintes disciplinas: Engenharia e Projeto de Software, Projeto Orientado a Objetos, Programação Para Web II e Seminários V.

Orientadores:

Prof. Marco Ikuro Hisatomi  
Prof. Paulo Henrique Terra  
Prof. Anderson Emidio de Macedo Gonçalves  
Prof. Roberto Yukio Nishimura

Uberlândia  
2017

## SUMÁRIO

1	INTRODUÇÃO .....	3
2	OBJETIVO .....	4
3	DESENVOLVIMENTO .....	5
3.1	ESCOPO .....	5
3.1.1	ESCOPO ORGANIZACIONAL.....	5
3.1.2	ESCOPO ESTRATÉGICO .....	6
3.1.3	ESCOPO TEMPORAL.....	6
3.2	JUSTIFICATIVA .....	7
3.3	Mapa MENTAL .....	8
3.4	REQUISITOS .....	9
3.4.1	REQUISITOS FUNCIONAIS.....	9
3.4.2	REQUISITOS NÃO FUNCIONAIS .....	9
3.5	CRONOGRAMA .....	10
3.6	METODOLOGIA DE DESENVOLVIMENTO E CICLO DE VIDA.....	11
3.7	GERENCIAMENTO DE QUALIDADE .....	13
3.7.1	PADRÕES DE PROGRAMAÇÃO E DOCUMENTAÇÃO.....	13
3.7.2	TESTES .....	14
3.7.2.1	RESPONSABILIDADES DO DESENVOLVEDOR.....	14
3.7.2.2	RESPONSABILIDADES DO ANALISTA DE TESTE/QUALIDADE.....	14
3.8	DEFINIÇÃO DE ARQUITETURA .....	15
3.8.1	LÓGICA .....	15
3.8.2	FÍSICA .....	16
3.8.2.1	PROTOCOLO SEM ESTADO.....	17
3.9	FRAMEWORKS .....	18
3.9.1	SPRING FRAMEWORK.....	18
3.9.2	ANGULARJS .....	19
3.10	CASOS DE USO .....	20
3.11	DIAGRAMA DE CLASSES .....	21
3.12	Modelagem de banco de dados .....	22
3.12.1	Conceitual .....	22
3.12.2	Lógica .....	23
3.13	CLASSES DE MODELO (Entidades Backend) .....	24

3.14	SERVIÇOS DE CONSULTA JAVASCRIPT ECMASCRIPT 6 (FRONT-END)	33
3.15	IMPLEMENTAÇÃO DO BANCO DE DADOS .....	36
3.15.1	DIAGRAMA .....	36
3.15.2	DDL .....	37
3.15.2.1	CIDADE .....	37
3.15.2.2	CLIENTE .....	37
3.15.2.3	ESTADO .....	38
3.15.2.4	ITEM DO PEDIDO .....	38
3.15.2.5	PASSO DA ENTREGA .....	38
3.15.2.6	PEDIDO .....	39
3.15.2.7	PRODUTO .....	39
3.16	TELAS DO SISTEMA .....	40
3.16.1	CONTROLE DE PRODUTOS .....	40
3.16.2	CONTROLE DE CLIENTES .....	41
3.16.3	CRIAÇÃO DE PEDIDO .....	43
3.16.4	LISTAGEM DE PEDIDOS .....	44
3.16.5	CONTROLE DE ENTREGA .....	45
3.16.6	MONITORAMENTO DE PEDIDO .....	46
4	CONCLUSÃO .....	47
	REFERÊNCIAS .....	48

## **1 INTRODUÇÃO**

O presente trabalho tem por finalidade o desenvolvimento textual e apresentação de artefatos técnicos construídos sob o eixo temático “Desenvolvimento de Sistemas II”. Para isso, vamos elaborar um projeto de software e analisar todas as fases executadas durante sua construção.

## **2 OBJETIVO**

O objetivo desta produção textual é apresentar uma análise completa do processo de desenvolvimento de um software para uma loja virtual. Serão apresentados todos os passos que levam à produção do artefato usável pelo cliente, bem como as justificativas para as tecnologias e ferramentas utilizadas.

Serão oferecidas propostas de soluções para o estudo de caso da loja virtual “Cacau Show”, onde a empresa necessita de um sistema web para que seus franqueados possam cuidar de seus clientes e gerenciar os pedidos efetuados.

### 3 DESENVOLVIMENTO

O desenvolvimento deste estudo se inicia pela análise do domínio do problema, onde o estudo de caso da loja virtual do cenário “Venda de Chocolates pela Internet” é feito sob a ótica do conjunto de disciplinas envolvidas no esforço do desenvolvimento do software.

#### 3.1 ESCOPO

O software desenvolvido será um aplicativo web acessível via navegadores modernos por franqueados da rede de venda de chocolates.

O sistema deverá manter os dados criados pelos usuários por tempo indeterminado e auxiliar no controle da integridade relacional dos dados inseridos, realizando os cálculos necessários de acordo com as ações efetuadas pelo usuário, como a remoção de produtos do estoque quando houver um pedido.

O aplicativo será acessado por franqueados em qualquer local do Brasil para gerenciar os dados de seus clientes, produtos, pedidos e estado em que cada uma destas entidades se encontram. O sistema não contará com interface administrativa para a rede, pois cada franquia terá seu próprio acesso com permissões totais na primeira versão do produto.

##### 3.1.1 ESCOPO ORGANIZACIONAL

O aplicativo será acessado por franqueados em qualquer local do Brasil para gerenciar os dados de seus clientes, produtos, pedidos e estado em que cada uma destas entidades se encontra. O sistema não contará com interface administrativa para a rede, pois cada franquia terá seu próprio acesso com permissões totais na primeira versão do produto.

### 3.1.2 ESCOPO ESTRATÉGICO

O sistema deve fornecer para o franqueado as ferramentas necessárias para que possa armazenar os dados de seus produtos, entregas, pedidos e clientes, como também deve fornecer recursos que favoreçam a agilidade no manuseamento dos elementos visuais apresentados pelo sistema.

### 3.1.3 ESCOPO TEMPORAL

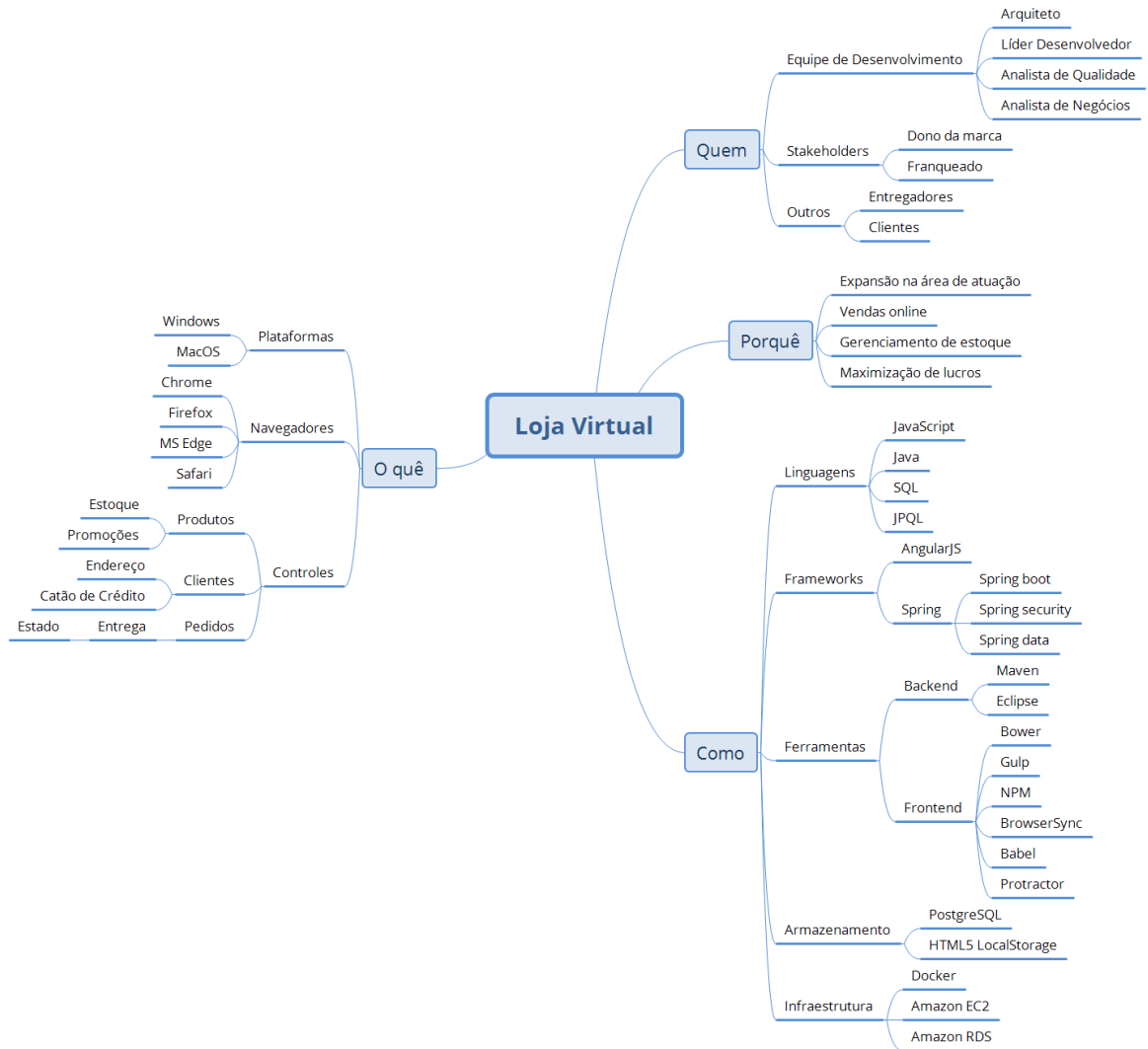
O sistema deverá ser utilizado a partir de sua implantação até o tempo definido, tendo como mínimo de expectativa de continuidade de manutenções evolutiva o período de quatro anos.



### 3.2 JUSTIFICATIVA

O sistema trará vantagens competitivas indispensáveis para o usuário final, sem as quais seria dificultoso manter os dados relevantes para a organização e de baixa confiabilidade a manutenção da integridade destes dados por interferência puramente humana.

### 3.3 MAPA MENTAL



### 3.4 REQUISITOS

#### 3.4.1 REQUISITOS FUNCIONAIS

- RF01 – O usuário poderá incluir um produto com nome, descrição, peso, embalagem, código e linha do produto.
- RF02 – O usuário poderá incluir clientes com nome, cpf, endereço, e-mail, data de nascimento, cidade, estado, etc.
- RF03 – O usuário poderá selecionar um cliente para iniciar um pedido.
- RF04 – O usuário poderá incluir produtos ao pedido e definir a quantidade a ser comprada.
- RF05 – O usuário poderá remover produtos do pedido.
- RF06 – O sistema não permitirá que um mesmo produto seja incluído mais de uma vez na lista de itens.
- RF07 – O usuário poderá adicionar um cartão ao cadastro do cliente.
- RF08 – O sistema não permitirá que um usuário cadastre mais que 5 cartões para um cliente.
- RF09 – O usuário poderá cadastrar promoções no sistema, informando desconto e vinculando a um produto existente.
- RF10 – O sistema deverá sugerir a substituição de um dos cartões quando o usuário estiver adicionando o 6º cartão.
- RF11 – O sistema deverá permitir que um o franqueado adicione apenas um código de promoção ao pedido.

#### 3.4.2 REQUISITOS NÃO FUNCIONAIS

- RNF01 – O sistema deve disponibilizar todas suas telas funcionando corretamente nos navegadores Firefox 53, Chrome 58, Microsoft Edge 38 e Safari.
- RNF02 – O sistema deve funcionar corretamente em duas versões maiores (53 → 52 → 51) anteriores dos navegadores suportados no RNF01.
- RNF03 – O sistema deve funcionar corretamente nos sistemas operacionais Windows 10, Windows 8, Windows 8.1 e Safari.

### 3.5 CRONOGRAMA

O cronograma tem previsão para término do projeto em 10 semanas.

Semana 1	Modelagem do Negócio
Semana 2	Análise de Requisitos
Semana 2	Definição de arquitetura
Semana 2 até semana 8	Implementação, testes
Semana 9 até semana 10	Implantação da primeira versão, correções, ajustes
Semana 11	Implantação da versão final

### 3.6 METODOLOGIA DE DESENVOLVIMENTO E CICLO DE VIDA

A escolha da metodologia de desenvolvimento e do ciclo de vida deve levar em conta vários aspectos. Analisaremos alguns destes fatores.

- **Identificação dos Requisitos de Usuário:** Temos uma identificação clara dos requisitos de software, pois nosso software não é complexo e as regras de negócio são mínimas.
- **Familiaridade com a tecnologia:** As tecnologias escolhidas, tanto para documentação, modelagem, análise quanto para implementação são familiares para os desenvolvedores.
- **Sistema complexo:** O sistema tem complexidade baixa.
- **Tempo para entrega:** O tempo para entrega é curto.
- **Limite de custo:** O projeto terá poucos recursos para desenvolvedores.
- **Documentação:** O projeto terá pouca documentação, os principais artefatos de documentação serão diagramas que, pelas restrições de custo e tempo, podem deixar de ser atualizados durante o curso do projeto.

Fatores	Waterfall	V-Shaped	Prototipação Evolucionária	Espiral	Iterativa e Incremental	Agile
Identificação dos Requisitos de Usuário	Pobre	Pobre	Bom	Excelente	Bom	Excelente
Familiaridade com a tecnologia	Pobre	Pobre	Excelente	Excelente	Bom	Pobre
Sistema complexo	Bom	Bom	Excelente	Excelente	Bom	Pobre
Sistema confiável	Bom	Bom	Pobre	Excelente	Bom	Bom
Tempo para entrega	Pobre	Pobre	Bom	Pobre	Excelente	Excelente
Forte gerenciamento de projeto	Excelente	Excelente	Excelente	Excelente	Excelente	Excelente
Limitação de custo	Pobre	Pobre	Pobre	Pobre	Excelente	Excelente
Visibilidade dos Stakeholders	Bom	Bom	Excelente	Excelente	Bom	Excelente
Limitação de habilidades	Bom	Bom	Pobre	Pobre	Bom	Pobre
Documentação	Excelente	Excelente	Bom	Bom	Excelente	Pobre
Reusabilidade de componentes	Excelente	Excelente	Pobre	Pobre	Excelente	Pobre

Fica evidente que uma metodologia Agile (Ágil) é a mais indicada para o desenvolvimento do software.

### 3.7 GERENCIAMENTO DE QUALIDADE

Para garantir que o software tem qualidade para uso, algumas métricas serão definidas afim de evitar erros durante o desenvolvimento e uso do sistema em produção. Durante todo o ciclo de vida, as verificações de qualidade deverão ser executadas e questionadas quando falham, evitando que uma funcionalidade seja entregue sem estar devidamente testada.

#### 3.7.1 PADRÕES DE PROGRAMAÇÃO E DOCUMENTAÇÃO

Um software pode ser modificado por várias equipes de desenvolvimento durante seu ciclo de vida, principalmente na fase de sustentação ou manutenção. Por isso, é necessário garantir que o código esteja devidamente documentado e siga padrões de desenvolvimento. As seguintes ações podem ser tomadas:

- **Revisão de código:** Antes de um código ser incluído ao código principal ele deve ser revisto pelos líderes de desenvolvimento ou mesmo outra pessoa na equipe que não seja o autor
- **Verificadores de estilo:** Alguns plug-ins podem ser utilizados e configurados para que alertas sejam disparados para o desenvolvedor quando os estilos de código definidos não forem respeitados.
- **Integração contínua:** É possível que sistemas de integração contínua possam executar os testes escritos pelos desenvolvedores e impeçam que sejam colocados em produção caso falhem.

## 3.7.2 TESTES

### 3.7.2.1 RESPONSABILIDADES DO DESENVOLVEDOR

Para que o desenvolvedor possa garantir a qualidade do software, é necessário que desenvolva dois tipos de testes automatizados:

- **Unitários:** Testam cada componente do sistema individualmente.
- **Integração:** Testam a integração entre os componentes do sistema.
- **Ponta-a-ponta:** Testa de forma automatizada todos os componentes e participantes do software, simulando um usuário real.

### 3.7.2.2 RESPONSABILIDADES DO ANALISTA DE TESTE/QUALIDADE

A equipe de testes deve garantir que o software cumpra com os requisitos funcionais e não funcionais. Para tal:

- De posse da documentação, os testes devem ser realizados através da perspectiva do usuário verificando cada regra de negócio e requisitos funcionais.
- Devem ser executados de forma sistemática, onde será possível emitir um relatório de quais testes passaram, quais não passaram e a data em que foram realizados.

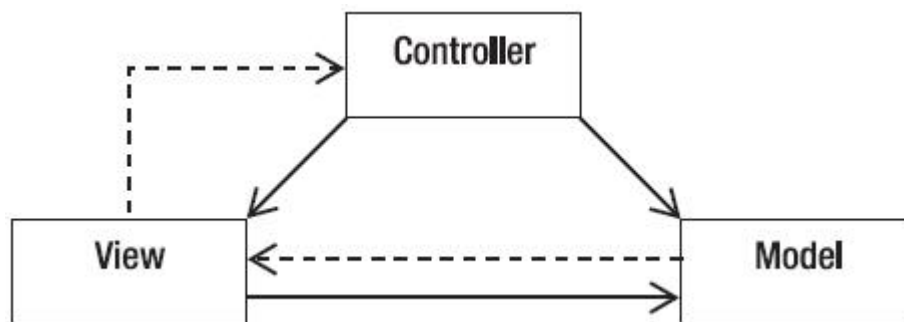


### 3.8 DEFINIÇÃO DE ARQUITETURA

#### 3.8.1 LÓGICA

O padrão MVC (Model-View-Controller) é um modelo de arquitetura de software em que a apresentação ou camada de comunicação com outros sistemas e/ou dispositivos (interface) é separada das camadas lógica (onde está o modelo do domínio) e da camada de acesso a dados. Também é chamada de arquitetura em 3 camadas.

No projeto desenvolvido, temos o padrão MVC no front-end e no back-end. A imagem abaixo ilustra a comunicação entre as três camadas:



### 3.8.2 FÍSICA

Nos sistemas distribuídos, o software de sistema é executado em um grupo de processadores fracamente integrados, que cooperam entre si conectados por uma rede.

Características dos sistemas distribuídos:

- **Compartilhamento de recursos.** Inclui recursos de hardware e software.
- **Abertura.** Sistemas podem ser ampliados utilizando recursos não proprietários a eles.
- **Concorrência.** Vários processos podem operar ao mesmo tempo em diferentes computadores na rede.
- **Escalabilidade.** São escalonáveis que significa que a capacidade do sistema pode ser aumentada por acréscimo de novos recursos.
- **Tolerância a defeitos.** Podem ser tolerantes a algumas falhas de hardware e software.
- **Transparência.** Usuário não precisa saber da natureza distribuída do sistema.

Desvantagens dos sistemas distribuídos:

- **Complexidade.** São mais complexos do que os sistemas centralizados.
- **Proteção.** É mais difícil gerenciar a proteção de um sistema distribuído.
- **Dificuldade de gerenciamento.** Maior esforço para gerenciamento.
- **Imprevisibilidade.** Podem ser imprevisíveis em suas respostas.

O desafio para os projetistas de sistemas distribuídos está em projetar software e hardware para que eles forneçam as características desejáveis dos sistemas distribuídos e ao mesmo tempo minimizem os problemas inerentes a esses sistemas. Questões relevantes de projetos de sistemas distribuídos são a identificação de recursos, comunicação, qualidade do serviço e a arquitetura do software.

Um sistema distribuído exige normalmente um software que gerencie as diversas partes e garanta que elas se comuniquem e troquem dados. Os sistemas distribuídos podem ser implementados com diferentes linguagens de programação, executados em processadores diferentes e podem utilizar diferentes protocolos de comunicação. O software de gerenciamento, conhecido por middleware, é uma espécie de intermediário dos diferentes componentes distribuídos do sistema.

### 3.8.2.1 PROTOCOLO SEM ESTADO

Protocolo sem estado, do inglês *stateless*, é um protocolo de comunicação onde cada par de requisições e respostas não tem conhecimento e não depende de requisições anteriores. Sendo assim, não é necessário que o servidor mantenha dados sobre a sessão do usuário conectado.

O uso de um protocolo e design *stateless* simplifica o desenvolvimento da aplicação e permite alta escalabilidade para sistemas distribuídos. Através de APIs RESTfuls expostas pelo servidor e consumidas por um client RESTful, é possível que o client obtenha exatamente o mesmo resultado para uma mesma requisição feita em servidores alojados em máquinas diferentes.

## 3.9 FRAMEWORKS

### 3.9.1 SPRING FRAMEWORK

O Spring Framework provê facilidades na programação e configuração de aplicações Java empresariais modernas em qualquer tipo de plataforma. O Spring fornece vários subprojetos plugáveis em seu “núcleo” para diversas preocupações comuns a aplicações web – segurança, autenticação, controle de transações, comunicação com banco de dados, tratamento de exceções, etc. Desta forma, estimula melhores design de código e fornece a base tecnológica para que o programador se concentre nas regras de negócio.

Principais características:

- Injeção de dependências
- Programação orientada a aspectos
- Mapeamento objeto-relacional com JPA
- Abstração de banco de dados
- Padrão MVC para web

O sistema desenvolvido neste estudo depende inteiramente do Spring Framework e utiliza de vários de seus recursos para desenvolvimento rápido da aplicação. Os principais recursos aproveitados foram:

- Java Persistence API para mapeamento objeto-relacional
- Spring Data JPA para reduzir a necessidade de codificação manual de operações CRUD
- Spring Events para uma abordagem mais orientada a eventos, reduzindo o acoplamento entre classes

### 3.9.2 ANGULARJS

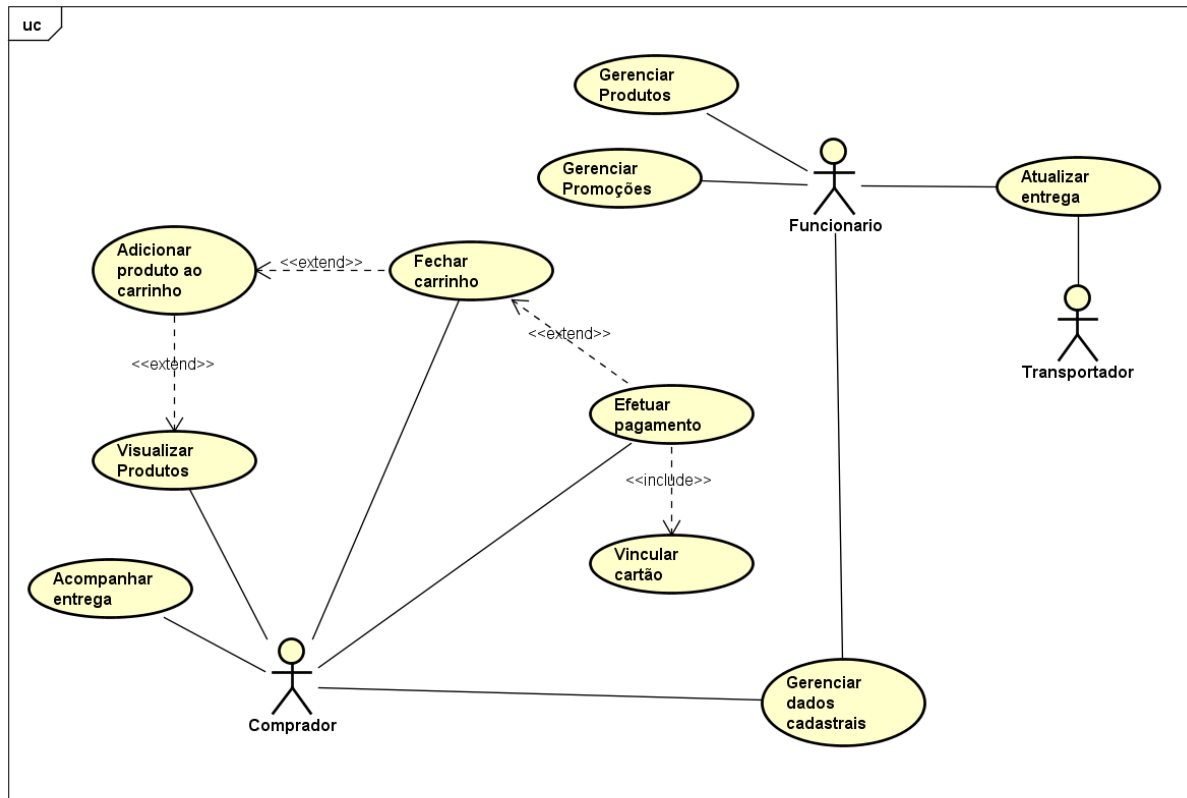
Para consumir a API RESTful exposta pelo back-end desenvolvido com Spring Framework, foi escolhido o framework front-end AngularJS.

O AngularJS é um framework JavaScript open-source mantido pelo Google que permite a criação de aplicações single-page de forma fácil e opinada. Seu modelo de arquitetura é o MVVM (model-view-view-model).

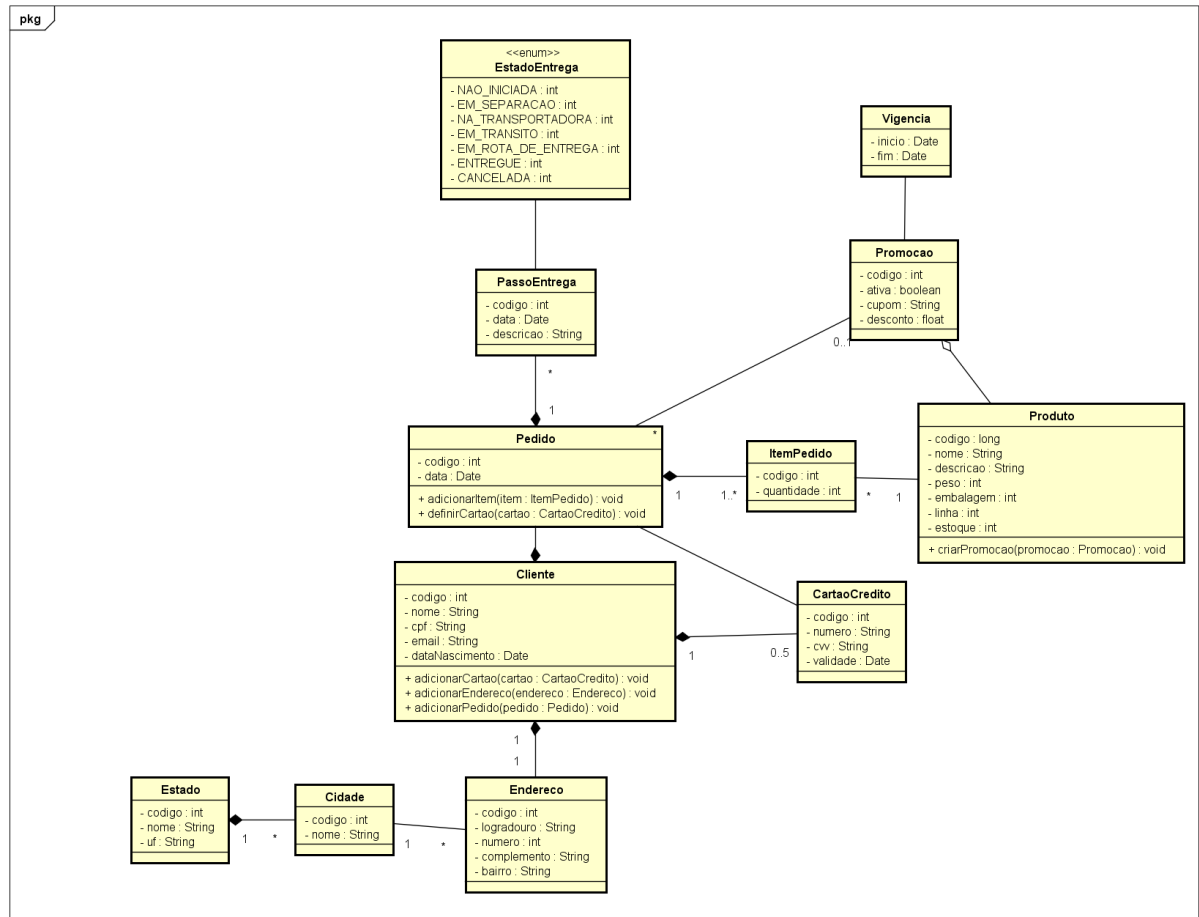
Suas principais características são:

- Injeção de dependências
- Foco em testes
- Permite que o desenvolvimento seja feito independentemente do back-end
- Opinado, incentiva criação de código usando padrões reconhecidos por outros desenvolvedores e melhora a qualidade de código

## 3.10 CASOS DE USO

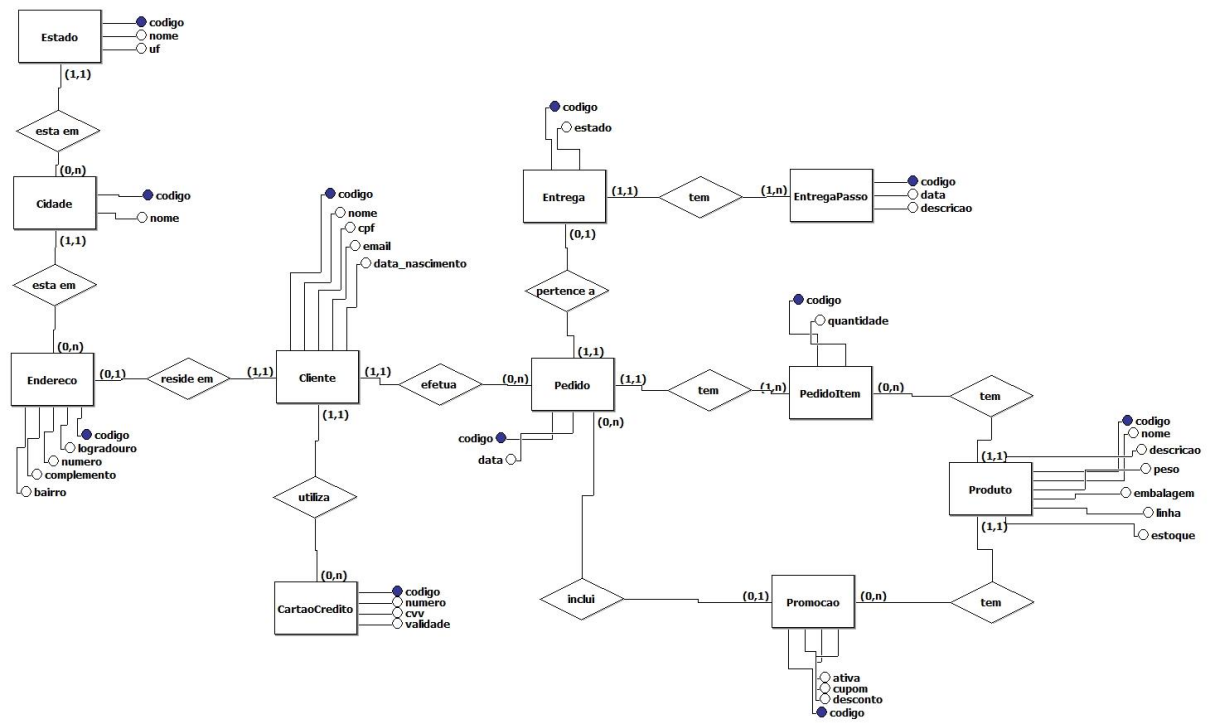


## 3.11 DIAGRAMA DE CLASSES



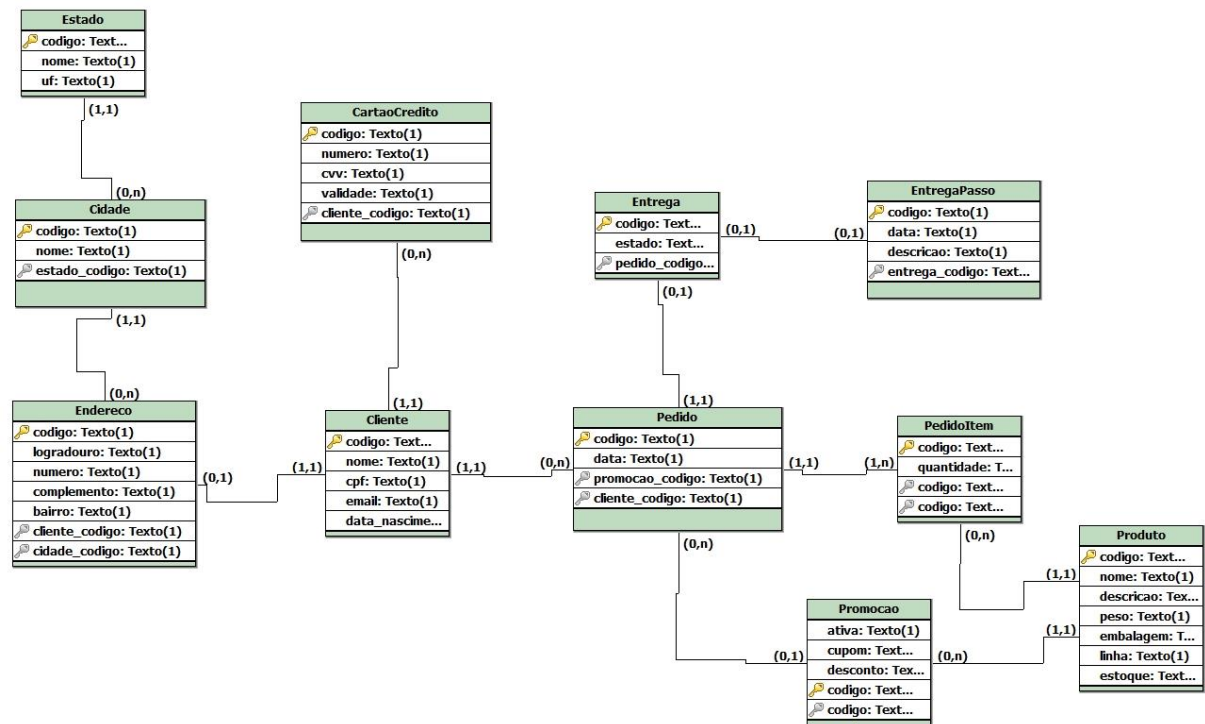
## 3.12 MODELAGEM DE BANCO DE DADOS

### 3.12.1 Conceitual





## 3.12.2 Lógica



### 3.13 CLASSES DE MODELO (ENTIDADES BACKEND)

As classes de modelo da aplicação são demonstradas a seguir. Todas as classes são entidades mapeadas com Java Persistence API e usam anotações do Projeto Lombok para geração de Setters e Getters automaticamente.

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.validation.constraints.NotNull;

import lombok.Data;

@Entity
@Data
public class Cidade {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @NotNull
    private String nome;

    @ManyToOne
    @NotNull
    private Estado estado;

    @Override
    public String toString() {
        return "Cidade [codigo=" + codigo + ", nome=" + nome + "]";
    }
}
```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import java.util.Date;

import javax.persistence.Embedded;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.NotBlank;

import lombok.Data;

@Entity
@Data
public class Cliente {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @NotBlank
    private String nome;

    @NotBlank
    private String cpf;

    @NotBlank
    private String email;

    @NotNull
    @Temporal(TemporalType.DATE)
    private Date dataNascimento;

    @NotNull
    @Embedded
    private Endereco endereco;
}
```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import javax.persistence.Embeddable;
import javax.persistence.ManyToOne;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.NotBlank;

import lombok.Data;

@Embeddable
@Data
public class Endereco {

    @NotBlank
    private String bairro;

    @NotNull
    @ManyToOne
    private Cidade cidade;
}
```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.OneToMany;

import org.hibernate.validator.constraints.NotBlank;

import lombok.Data;

@Entity
@Data
public class Estado {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @NotBlank
    private String nome;

    @NotBlank
    private String uf;

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "estado")
    private List<Cidade> cidades = new ArrayList<>();
}
```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import lombok.AllArgsConstructor;
import lombok.Getter;

@Getter
@AllArgsConstructor
public enum EstadoEntrega {
    // @formatter:off
    NAO_INICIADA("Não iniciada"),
    EM_SEPARACAO("Em separação"),
    NA_TRANSPORTADORA("Na transportadora"),
    EM_TRANSITO("Em trânsito"),
    EM_ROTA_DE_ENTREGA("Em rota de entrega"),
    ENTREGUE("Entregue"),
    CANCELADA("Cancelada");
    // @formatter:on

    private String descricao;
}
```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.OneToOne;
import javax.validation.constraints.NotNull;

import lombok.Data;

@Entity
@Data
public class ItemPedido {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @NotNull
    private Integer quantidade;

    @OneToOne
    @NotNull
    private Produto produto;

    @ManyToOne
    @JoinColumn
    @NotNull
    private Pedido pedido;
}
```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.EnumType;
import javax.persistence.Enumerated;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

import lombok.Data;

@Entity
@Data
public class PassoEntrega {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @Temporal(TemporalType.TIMESTAMP)
    private Date data;

    private String descricao;

    @Enumerated(EnumType.STRING)
    private EstadoEntrega estado;

    @ManyToOne(optional=false)
    @JoinColumn
    private Pedido pedido;
}
```



```

package io.github.marcelothebuilder.unoparpedidos4.model;
import java.util.Date;
import java.util.List;
import javax.persistence.CascadeType;
import javax.persistence.Entity;
import javax.persistence.EntityListeners;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.ManyToOne;
import javax.persistence.OneToMany;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;
import javax.persistence.Transient;
import javax.validation.constraints.NotNull;
import io.github.marcelothebuilder.unoparpedidos4.service.PedidoService;
import lombok.Data;

@Entity
@EntityListeners(PedidoService.class)
@Data
public class Pedido {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @Temporal(TemporalType.TIMESTAMP)
    private Date data = new Date();

    @OneToMany(fetch=FetchType.EAGER, mappedBy="pedido",
        cascade={CascadeType.PERSIST, CascadeType.MERGE})
    private List<ItemPedido> itens;

    @ManyToOne(fetch=FetchType.EAGER)
    @NotNull
    private Cliente cliente;

    @OneToMany(fetch=FetchType.LAZY, mappedBy="pedido",
        cascade={CascadeType.PERSIST, CascadeType.MERGE})
    private List<PassoEntrega> passosEntrega;

    public void setItens(List<ItemPedido> itens) {
        this.itens = itens;
        this.itens.forEach(item -> item.setPedido(this));
    }

    @Transient
    public Integer getQuantidadeItens() {
        return getItens().size();
    }

    public void adicionarPassoEntrega(PassoEntrega passo) {
        passo.setPedido(this);
        this.getPassosEntrega().add(passo);
    }
}

```

```
package io.github.marcelothebuilder.unoparpedidos4.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.validation.constraints.NotNull;

import org.hibernate.validator.constraints.NotBlank;

import lombok.Data;

@Entity
@Data
public class Produto {
    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long codigo;

    @NotBlank
    private String nome;

    private String descricao;

    @NotNull
    private Integer peso;

    @NotNull
    private Integer estoque;
}
```

### 3.14 SERVIÇOS DE CONSULTA JAVASCRIPT ECMASCRIPT 6 (FRONT-END)

```
class PedidoService {
  constructor($http, $log) {
    angular.extend(this, {
      $http,
      $log
    });
  }

  porCodigo(codigo) {
    return this.$http.get(`/pedido/${codigo}`);
  }

  todos() {
    return this.$http.get('/pedido');
  }

  adicionar(pedido) {
    return this.$http.post('/pedido', pedido);
  }

  criar(pedido) {
    return this.adicionar(pedido);
  }

  adicionarPassoEntrega(codigoPedido, passo) {
    passo.pedidoCodigo = codigoPedido;
    return this.$http.post(`/pedido/${codigoPedido}/passo-entrega`, passo);
  }

  buscarPassos(codigoPedido) {
    return this.$http.get(`/pedido/${codigoPedido}/passo-entrega`);
  }
}

angular.module('app')
  .service('PedidoService', PedidoService);
```

```
class ProdutoService {
  constructor($http, $log) {
    angular.extend(this, {
      $http,
      $log
    });
  }

  todos() {
    return this.$http.get('/produto');
  }

  adicionar(produto) {
    return this.$http.post('/produto', produto);
  }

  porNome(nome, pagina = 0, tamanho = 10) {
    return this.$http.get('/produto', {
      params: {
        nome,
        page: pagina,
        size: tamanho
      }
    });
  }
}

angular.module('app')
  .service('ProdutoService', ProdutoService);
```

```
class ClienteService {
  constructor($http, $log) {
    angular.extend(this, {
      $http,
      $log
    });
  }

  porCodigo(codigo) {
    return this.$http.get(`/cliente/${codigo}`);
  }

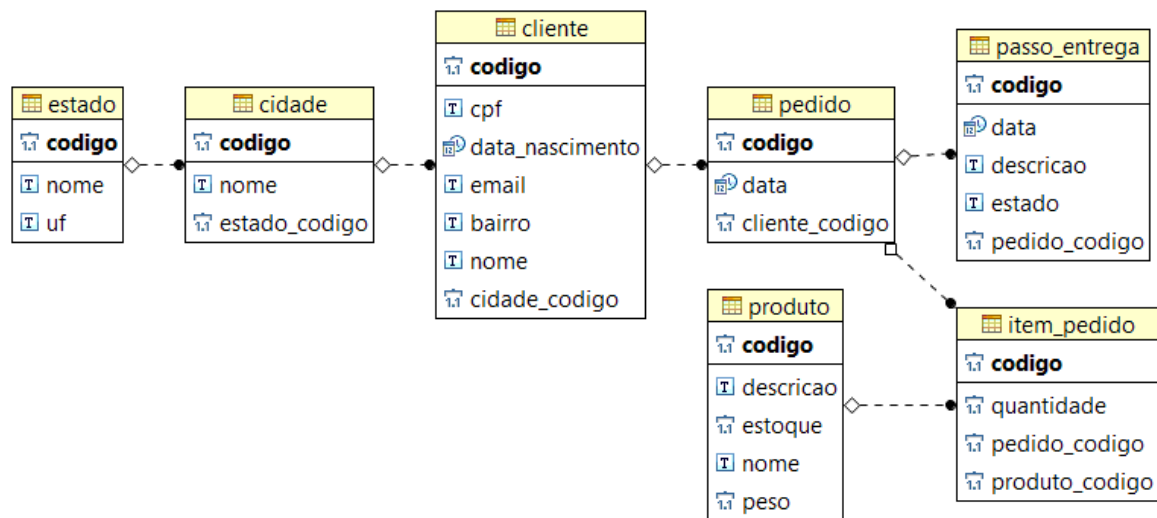
  todos() {
    return this.$http.get('/cliente');
  }

  adicionar(cliente) {
    return this.$http.post('/cliente', cliente);
  }
}

angular.module('app')
  .service('ClienteService', ClienteService);
```

### 3.15 IMPLEMENTAÇÃO DO BANCO DE DADOS

#### 3.15.1 DIAGRAMA



### 3.15.2 DDL

#### 3.15.2.1 CIDADE

```
CREATE TABLE public.cidade (  
    codigo int8 NOT NULL,  
    nome varchar(255) NOT NULL,  
    estado_codigo int8 NOT NULL,  
    CONSTRAINT cidade_pkey PRIMARY KEY (codigo),  
    CONSTRAINT fkbagibbbmm24rh38jgcl6vvc8a FOREIGN KEY  
(estado_codigo) REFERENCES public.estado(codigo)  
)
```

#### 3.15.2.2 CLIENTE

```
CREATE TABLE public.cliente (  
    codigo int8 NOT NULL,  
    cpf varchar(255) NOT NULL,  
    data_nascimento date NOT NULL,  
    email varchar(255) NOT NULL,  
    bairro varchar(255) NULL,  
    nome varchar(255) NOT NULL,  
    cidade_codigo int8 NULL,  
    CONSTRAINT cliente_pkey PRIMARY KEY (codigo),  
    CONSTRAINT fk2i7b0my9m83qtha2ws54yb1su FOREIGN KEY  
(cidade_codigo) REFERENCES public.cidade(codigo)  
)
```

### 3.15.2.3 ESTADO

```
CREATE TABLE public.estado (  
    codigo int8 NOT NULL,  
    nome varchar(255) NOT NULL,  
    uf varchar(255) NOT NULL,  
    CONSTRAINT estado_pkey PRIMARY KEY (codigo)  
)
```

### 3.15.2.4 ITEM DO PEDIDO

```
CREATE TABLE public.item_pedido (  
    codigo int8 NOT NULL,  
    quantidade int4 NOT NULL,  
    pedido_codigo int8 NOT NULL,  
    produto_codigo int8 NOT NULL,  
    CONSTRAINT item_pedido_pkey PRIMARY KEY (codigo),  
    CONSTRAINT fkf1hj32yci9witllvwlyjlmdo7 FOREIGN KEY  
(produto_codigo) REFERENCES public.produto(codigo),  
    CONSTRAINT fkfxhc6kl8ffjp9dsxjp2djhiut FOREIGN KEY  
(pedido_codigo) REFERENCES public.pedido(codigo)  
)
```

### 3.15.2.5 PASSO DA ENTREGA

```
CREATE TABLE public.passo_entrega (  
    codigo int8 NOT NULL,  
    "data" timestamp NULL,  
    descricao varchar(255) NULL,  
    estado varchar(255) NULL,  
    pedido_codigo int8 NOT NULL,  
    CONSTRAINT passo_entrega_pkey PRIMARY KEY (codigo),  
    CONSTRAINT fkldrom001dcyocudw0yffgc8ve FOREIGN KEY  
(pedido_codigo) REFERENCES public.pedido(codigo)  
)
```



### 3.15.2.6 PEDIDO

```
CREATE TABLE public.pedido (  
    codigo int8 NOT NULL,  
    "data" timestamp NULL,  
    cliente_codigo int8 NOT NULL,  
    CONSTRAINT pedido_pkey PRIMARY KEY (codigo),  
    CONSTRAINT fkily4j6ymnwg9wiff282uv17wl FOREIGN KEY  
(cliente_codigo) REFERENCES public.cliente(codigo)  
)
```

### 3.15.2.7 PRODUTO

```
CREATE TABLE public.produto (  
    codigo int8 NOT NULL,  
    descricao varchar(255) NULL,  
    estoque int4 NOT NULL,  
    nome varchar(255) NOT NULL,  
    peso int4 NOT NULL,  
    CONSTRAINT produto_pkey PRIMARY KEY (codigo)  
)
```

### 3.16 TELAS DO SISTEMA

#### 3.16.1 CONTROLE DE PRODUTOS

PEDIDOS

PRODUTOS

CLIENTES

MINHA CONTA

Nome \*

Chocolate com mel

Descrição

Chocolate com sabores de mel

29 / 150

Peso (Grams) \*

14

Estoque \*

121

Chocolate

Chocolate simples e barato

Peso: 1

Estoque: 12

Chocolate branco

Um delicioso chocolate

Peso: 12

Estoque: 12

+

## 3.16.2 CONTROLE DE CLIENTES

PEDIDOS

PRODUTOS

CLIENTES

MINHA CONTA

Nome \*

Marcelo

CPF \*

09121301291

Email \*

marcelo@email.com

Estado \*


Minas Gerais - MG

Cidade \*

Uberlândia

Bairro \*

Martins



8/1/1993


Christopher

chris@email.com

Dec 1, 1991

131231312321

Martins



Nome \*

CPF \*

Email \*

Estado \*

Nenhum



Cidade \*



Bairro \*



Data de nascimento

Christopher

chris@email.com

Dec 1, 1991

131231312321

Martins

Uberlândia - Minas Gerais

CRIAR PEDIDO

Marcelo

marcelo@email.com

Aug 1, 1993

09121301291

Martins

Uberlândia - Minas Gerais

CRIAR PEDIDO



## 3.16.3 CRIAÇÃO DE PEDIDO

PEDIDOS

PRODUTOS

CLIENTES

MINHA CONTA

Novo pedido para:

Marcelo

marcelo@email.com

Aug 1, 1993

09121301291

Martins

Uberlândia - Minas Gerais

Selecione um produto \*

Quantidade \*

+

Chocolate branco

Quantidade: 12

Chocolate com mel

Quantidade: 1

### 3.16.4 LISTAGEM DE PEDIDOS

The screenshot displays a web application interface. At the top, a dark blue header bar contains three navigation links: 'PEDIDOS' (highlighted with a light blue background), 'PRODUTOS', and 'CLIENTES'. On the right side of this header is a red button labeled 'MINHA CONTA'. Below the header, on the left, is a light gray sidebar. Inside the sidebar, a white box contains the following information: 'Cliente: Marcelo', 'Data: Jun 6, 2017', and 'Itens: 2'. The main content area to the right of the sidebar is currently empty.

## 3.16.5 CONTROLE DE ENTREGA

### Cliente

**Marcelo**

marcelo@email.com

Aug 1, 1993

09121301291

Martins

Uberlândia - Minas C

### Pedido

Data: Jun 6, 2017

Itens: 2

Em separação

Descrição

pedido em separação para entrega

32 / 150

CANCELAR

ADICIONAR

+

### Entrega

### 3.16.6 MONITORAMENTO DE PEDIDO

Marcelo

marcelo@email.com

Aug 1, 1993

09121301291

Martins

Uberlândia - Minas Gerais

Pedido

Data: Jun 6, 2017

Itens: 2

Entrega

Jun 6, 2017 - Em separação : pedido em separação para entrega



## 4 CONCLUSÃO

Com o estudo e pesquisas realizadas sobre os temas abordados neste trabalho, foi possível praticar e aplicar os conhecimentos adquiridos nas disciplinas de Engenharia e Projeto de Software, Projeto Orientado a Objetos, Programação para Web II e Seminário V.

Foi possível demonstrar como a documentação, definição de arquitetura, cronograma, mapas mentais e diagramas UML podem auxiliar no desenvolvimento de um software.

A produção do sistema também foi satisfatória, onde foi de extrema importância utilizar frameworks modernos para atingir os objetivos de escalabilidade, distributividade e rapidez no desenvolvimento.

Em geral, é possível afirmar que as pesquisas realizadas agregaram conhecimento para o pesquisador e o material produzido oferece de forma clara referências para a produção de um software em cada uma de suas etapas.

## REFERÊNCIAS

UML. **Wikipédia**. Disponível em: <<https://pt.wikipedia.org/wiki/UML>>. Acesso em: quarta-feira, 6 de junho de 2017.

Protocolo sem estado. **Wikipédia**. Disponível em: <[https://pt.wikipedia.org/wiki/Protocolo\\_sem\\_estado](https://pt.wikipedia.org/wiki/Protocolo_sem_estado)>. Acesso em: quarta-feira, 6 de junho de 2017.

Sommerville, Ian, "Engenharia de Software", 9ª ed, São Paulo, Pearson Prentice Hall, 2011, ISBN 978-85-7936-108-1.

MACEDO, D. Arquitetura de Aplicações. **Um pouco de tudo sobre TI**. Disponível em: <<http://www.diegomacedo.com.br/arquitetura-de-aplicacoes-em-2-3-4-ou-ncamadas/>>. Acesso em: quarta-feira, 6 de junho de 2017.

ARQUITETURA de dados. **Wikipedia**. Disponível em <[https://pt.wikipedia.org/wiki/Arquitetura\\_de\\_dados](https://pt.wikipedia.org/wiki/Arquitetura_de_dados)>. Acesso em: quarta-feira, 6 de junho de 2017.