CS5500 Fall 2019

Pizza Paradise API Design Document

Team members:

Name	Email
Jada Greene	greene.jad@husky.neu.edu
Yejee (Jenny) Lee	lee.yej@husky.neu.edu
Clara Mae Wells	wells.cl@husky.neu.edu

TABLE OF CONTENTS

1. Introduction

The project is to create a RESTful API system that supports a pizza ordering website for the Pizza Paradise Company. The system fetches a record or set of resources from the database, creates a new resource or set of resources, updates or replaces a given record, and deletes a given resource.

This design document presents the designs to be used in implementing the project. The designs described follow the requirements specified in the Greene Lee Wells Decision Document titled “Pizza Paradise Policies.”

1.1. Purpose

The purpose of this document is to present a detailed description of the designs of the RESTful API system created to support the pizza ordering website for Pizza Paradise. This document is intended for the members of the design team to use as a guideline to implement the project. The document is also intended for designers who may upgrade or modify the present design.

1.2. Scope

This document provides a comprehensive description of the software architecture of the RESTful API system. It describes the structure and design of the various APIs that call specified information.

1.3. References

The user of this design document may need the following documents for reference:

- Greene Lee Wells Decision Document titled “Pizza Paradise Policies”
- Greene Lee Wells Lucid Chart titled “Pizza API UML” - Lucid Chart - Pizza API UML

1.4. Overview

The Pizza Paradise Design Document is divided into 6 sections, the sections are:

1. Introduction
2. Design considerations
3. Architecture
4. Data design
5. Software interface design
6. Appendices

2. Design considerations

2.1. Assumptions

This document assumes the user has general knowledge of Java. The user has login access to MongoDB and Heroku technologies.

2.2 Constraints

The system is dependent on MongoDB database to pull data from. The system is implemented using Java, MongoDB, and Heroku technologies.

2.3 System environment

The web based RESTful API pizza ordering system is designed to work on all operating systems. The system is accessible through any laptop, desktop, and smartphone connected to an internet service provider. The system is accessible at all times.

2.4. Design methodology

The system is designed with flexibility for future development and/or modification.

2.4.1 Goals

- Provide RESTful API system to service pizza ordering website.

2.4.2 Non-Goals

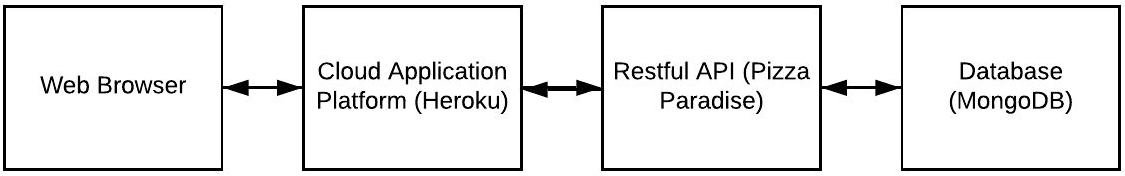
The system does not aim to do the following:

- Collect customer information (e.g. name and address)
- Collect payment information or process payments
- Offer support for non-delivery order sources, such as call-in orders
- Offer option for pickup vs delivery order
- Offer a mobile app
- Offer order tracking

3. Architecture

3.1. System design

The diagram below shows the principal parts of the RESTful API pizza ordering system and their interactions.



4. Data design

4.1. Data description

MongoDB database and Spring Framework to communicate with the database that is made accessible at all times through Heroku.

4.2 Data dictionary

The project is linked to the MongoDB database titled “pizza-paradise.” The table below describes the collections within the pizza-paradise database.

Collection Name	**Field Name**	**Type**
Cart	_id	ObjectId
	pizzas	Array
	sides	Array

	storeID	String
	totalAmount	String
	specialApplied	boolean
	_class	Cart
PizzaSize	_id	String
	sizeName	String
	sizeInch	String
	price	Double
SideItem	_id	String
	name	String
	price	Double
	type	String
SpecialItem	_id	String
	name	String
	description	String
StoreItem	_id	String
	streetNumandName	String
	city	String
	state	String
	zipCode	String
	offersGlutenFree	Boolean
ToppingItem	_id	String
	toppingName	String
	toppingType	String

	toppingGluten	String
	toppingSmallPrice	Double
	toppingMediumPrice	Double
	toppingLargePrice	Double
Receipt	_id	ObjectId
	timePlaced	Object
	cart	Object
	card	Object
	_class	Receipt

5. Software interface design

5.1 Model description

Descriptions of models are presented in the tables below.

PizzaSize (model) Creates new PizzaSize given id, sizeName, sizeInches, and price	
Attributes	**Description**
private String id	The id of the pizza size
private String sizeName	The name of the pizza size
private String sizeInch	The size of the pizza in inches
private Double price	The price of the pizza size
Operations	**Description**
get() functions: ● public String getId() ● public String getSizeName() ● public String getSizeInch() ● public Double getPrice()	
set() functions: ● public void setId(String id) ● public void setSizeName(String sizeName) ● public void setSizeInch(String sizeInch)	

• public void setPrice(Double price)	
public String toString()	Return string representation of PizzaSize (id + sizeName + sizeInch + price)
public boolean equals(Object obj)	Checks if two objects are equal

SideItem (model) Creates new side item given id, name, price, and type	
Attributes	**Description**
private String id	The id of the side item
private String name	The name of the side item
private Double sizeInch	The side item price
Operations	**Description**
get() functions: • public String getId() • public String getName() • public Double getPrice() • public String getType()	
set() functions: • public void setId(String id) • public void setName(String name) • public void setPrice(Double price) • Public void setType(String type)	
public String toString()	Return string representation of SideItem (id + name + price + type)
public int hashCode()	Return hash code value of SideItem
public boolean equals(Object obj)	Checks if two objects are equal

SpecialItem (model) Creates new special item given id, name, and description	
Attributes	**Description**
private String id	The id of the special item
private String name	The name of the special item
private String description	The description of special
Operations	**Description**

get() functions: ● public String getId() ● public String getName() ● public String getDescription()	
set() functions: ● public void setId(String id) ● public void setName(String name) ● public void setDescription(String description)	
public String toString()	Return string representation of SpecialItem (id + name + description)
public boolean equals(Object obj)	Checks if two objects are equal

StoreItem (model) Creates new store item given id, streetNumAndName, city, state, and zipCode	
Attributes	**Description**
private String id	The id of the store item
private String streetNumAndName	The street number and street name of store item
private String city	The city of the store item
private String state	The state of the store item
private String zipcode	The zip code of the store item
Operations	**Description**
get() functions: ● public String getId() ● public String getStreetNumAndName() ● public String getCity() ● public String getState() ● public String getZipCode()	
set() functions: ● public void setId(String id) ● public void setstreetNumAndName(String streetNumAndName) ● public void setCity(String city) ● public void setState(String state) ● public void setZipCode(String zipCode)	
public String toString()	Return string representation of StoreItem (id + streetNumAndName + city + state + zipCode)

public int hashCode()	Return hash code value for StoreItem
public boolean equals(Object obj)	Checks if two objects are equal

ToppingItem (model) Creates new topping item given id, toppingName, toppingType, toppingSmallPrice, toppingMediumPrice, toppingLargePrice, and toppingGluten	
Attributes	**Description**
private String id	The id of the topping item
private String toppingName	The name of the topping
private String toppingType	The topping type
private Double toppingSmallPrice	The price of a small topping
private Double toppingMediumPrice	The price of a medium topping
private Double toppingLargePrice	The price of a large topping
private String toppingGluten	The topping gluten or non-gluten
Operations	**Description**
get() functions: ● public String getId() ● public String getToppingName) ● public String getToppingType() ● public Double getToppingSmallPrice() ● public Double getToppingMediumPrice() ● public Double getToppingLargePrice() ● public String getToppingGluten()	
set() functions: ● public void setId(String id) ● public void setToppingName(String toppingName) ● public void setToppingType(String toppingType) ● public void setToppingSmallPrice(Double toppingSmallPrice) ● public void setToppingMediumPrice(Double toppingMediumPrice) ● public void setToppingLargePrice(Double toppingLargePrice) ● Public void setToppingGluten	
public String toString()	Return string representation of Toppingtem (id + toppingName + toppingType + toppingSmallPrice + toppingMediumPrice + toppingLargePrice + toppingGluten)

public boolean equals(Object obj)	Checks if two objects are equal.

Pizza (model) Creates new Pizza given sizeID, gluten	
Attributes	**Description**
private String sizeID	The sizeID of this Pizza
private boolean gluten	True = gluten, False = glutenFree
private List<String> toppingIDs	The list of toppingIDs in this Pizza
private int MAX_TOPPING = 4	The maximum number of toppings of this Pizza
Operations	**Description**
get() functions: ● public String getSizeID() ● public List<String> getToppingIDs() ● public boolean isGluten()	
public String toString()	Return string representation of Pizza (sizeId + gluten + toppingIDs)
public boolean equals(Object obj)	Checks if two objects are equal

Cart (model) Creates new Cart given storeId, cartId	
Attributes	**Description**
private String storeId	The storeId of this Cart
private ObjectId id	The cartId of this Cart
private List<Pizza> pizzas	The list of Pizza object in this Cart
private List<String> sides	The list of sideIds in this Cart
private Double totalAmount	The total amount of this Cart
Private boolean specialApplied	True = specialApplied / False = specialNotAplied
Operations	**Description**
get() functions: ● public String getId() ● public String getStoreID()	

• public Double getTotalAmount() • public List<Pizza> getPizzas() • public List<String> getSides() • public boolean isSpecialApplied()	
set() functions: • public void setId(ObjectId id) • public void setStoreID(String id) • public void setTotalAmount(Double totalAmount) • public void setSpecialApplied(boolean applied)	
public String toString()	Return string representation of Cart (cartId + storeId + list of pizzas + list of sides + total price + specialApplied)
public boolean equals(Object obj)	Checks if two objects are equal

Card (model) Creates new Card given firstName, lastName, cardNumber, expMonth, expYear, cardProvider	
Attributes	**Description**
private String firstName	The first name of a Card holder
private String lastName	The last name of a Card holder
private String cardNumber	The card number of a Card
private Integer expMonth	The expiration month of a Card
private Integer expYear	The expiration year of a Card
private CardProvider provider	The card provider of a Card
Operations	**Description**
get() functions: • public String getFirstName() • public String getLastName() • public String getCardNumber() • public Integer getExpMonth() • public Integer getExpYear() • public CardProvider getCardProvider()	
set() functions: • public void setCardNumber(String securedNum)	Set the card number with only 4 digits after purchase
public String toString()	Return string representation of Card (firstName +

	lastName + cardProvider+ cardNumber + expMonth + expYear)
public boolean equals(Object obj)	Checks if two objects are equal

Receipt (model) Creates new Receipt given cart and card	
Attributes	**Description**
private ObjectId receiptId	The id of a Receipt
private GregorianCalendar timePlaced	The time this Receipt was created
private Cart cart	The cart that was purchased on the Receipt
private Card card	The card that was used as a payment on the Receipt
Operations	**Description**
get() functions: ● public String getReceiptId() ● public GregorianCalendar getTimePlaced() ● public Cart getCart() ● public Card getCard()	
public String toString()	Return string representation of Receipt (receiptId + timePlaced + pizzas + sides + totalAmount + cardNumber(4digits))
public boolean equals(Object obj)	Checks if two objects are equal

5.2 Response Object description

CartAddResponse (model) Creates new Response when a customer add an item to the Cart Also constructs a failing response given a message	
Attributes	**Description**
private boolean success	The success of adding items to Cart
private Pizza pizza	The Pizza added to Cart
private SideItem side	The sideItem added to Cart
Private String cartID	The cartID that was assigned to Cart

Private String storeID	The storetID that was assigned to Cart
Private String message	The optional message regarding adding to Cart
Operations	**Description**
get() functions: ● public boolean getSuccess() ● public Pizza getPizza() ● public SideItem getSide() ● public String getCartID() ● public String getStoreID() ● public String getMessage()	
public String toString()	Return string representation of CardAddResponse (success + cartID + storeID + pizza + side + message)
public boolean equals(Object obj)	Checks if two objects are equal

PriceResponse (model) Creates new priceResponse when get total price of a Cart given a cartId and StoreId Also constructs a failing response given a message	
Attributes	**Description**
private boolean success	The success of getting price
private Double price	The price found
private String currency	The currency used for price
private String message	The message regarding price
Operations	**Description**
get() functions: ● public boolean isSuccess() ● public String getCurrency() ● public Double getPrice() ● public String getMessage()	

ApplySpecialResponse (model) Creates new response given a specialId and savings Also constructs a failing response given a message	
Attributes	**Description**
private String specialId	The id of the special item

private boolean success	The success of applying special
private String message	The message regarding applying special
private Double savings	The savings received from special
Operations	**Description**
get() functions: ● public String getSpecialId() ● public boolean getSuccess() ● public String getMessage() ● public Double getSavings()	
set() functions: ● public void setSpecialId(String specialId) ● public void setSuccess(boolean success) ● public void setMessage(String message) ● public void setSavings(Double savings)	
public String toString()	Return string representation of StoreItem (id + streetNumAndName + city + state + zipCode)
Public int hashCode()	Return hash code value for ApplySpecial
public boolean equals(Object obj)	Checks if two objects are equal

PurchaseResponse (model) Creates new PurchaseResponse when a customer makes a purchase using cartID, storeID and Card Also constructs a failing response given a message	
Attributes	**Description**
private boolean success	The success of making purchase
private Receipt receipt	The receipt found
private String message	The message regarding Purchase
Operations	**Description**
get() functions: ● public boolean isSuccess() ● public Receipt getReceipt() ● public String getMessage()	

PizzaSuggestionResponse (model)

Creates new PizzaSuggestionResponse that suggests the number of small, medium, or large pizzas.	
Attributes	**Description**
private Integer small	The number of small pizzas suggested
private Integer medium	The number of medium pizzas suggested
private Integer large	The number of large pizzas suggested
Operations	**Description**
get() functions: ● public Integer getSmall() ● public Integer getMedium() ● public Integer getLarge()	

5.3 API description

The system consists of multiple APIs to be called to return specific information requested for pizza delivery. Descriptions of APIs are presented in the tables below. See section 5.1 for reference.

To access pizza-paradise API visit, https://pizza-paradise.herokuapp.com/

PizzaSizeApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/size**	
Use case: A pizza store serves more than one size pizza or serves sizes that are uncommon	
API Operations	**Description**
GET/size	Gets a list of all PizzaSize
GET/size/{id}	Gets a specific PizzaSize by id
POST/size	Adds a PizzaSize to pizza-paradise database
DELETE/size/{id}	Deletes a specific PizzaSize by id from the pizza-paradise database

SpecialApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/special**	
Use case: A grand opening or other promotion prompts store owner to offer specials	
API Operations	**Description**
GET/special	Gets a list of all SpecialItems
GET/special/{id}	Gets a specific SpecialItem by id
POST/special/add	Adds a SpecialItem to pizza-paradise database

DELETE/special/delete{id}	Deletes a specific SpecialItem by id from the pizza-paradise database

StoreApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/store**	
Use case: A pizza store has multiple stores or has expansion plans	
API Operations	**Description**
GET/store	Return list of all StoreItem
GET/store/{id}	Return a specific StoreItem by id
POST/store/add	Adds a StoreItem to pizza-paradise database
DELETE/store/delete/{id}	Deletes a specific StoreItem by id from the pizza-paradise database

SideApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/side**	
Use case: A pizza store sells items in addition to pizza	
API Operations	**Description**
GET/side	Gets a list of all SideItem
GET/side/{id}	Gets a specific SideItem by id
POST/side/add	Adds a SideItem to pizza-paradise database
DELETE/side/delete/{id}	Deletes a specific SideItem by id from the pizza-paradise database

ToppingApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/topping**	
Use case: A pizza store offers toppings on pizza rather than a classic margherita	
API Operations	**Description**
GET/topping	Gets a list of all ToppingItem
GET/topping/{id}	Gets a specific ToppingItem by id

POST/topping/add	Adds a ToppingItem to pizza-paradise database
DELETE/topping/delete/{id}	Deletes a specific ToppingItem by id from the pizza-paradise database

CartApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/cart**	
Use case: Desire to put a pizza together and view list and price of items to order	
API Operations	**Description**
GET/cart/{storeId}/{cartId}	Gets a list of all items in the Cart
GET/cart/{storeId}/{cartId}/price	Return a list of all prices of items in a specific Cart
POST/cart/{storeId}/{cartId}/pizza	Adds a pizza object to a specific Cart
POST/cart/{storeId}/{cartId}/side	Adds a sideId to a Cart
DELETE/cart/{storeId}/{cartId}	Deletes a specific Cart
DELETE/cart/{storeId}/{cartId}/side	Deletes a SideItem from a Cart
DELETE/cart/{storeId}/{cartId}/pizza	Deletes a pizza object from a specific Cart
*Pizza Object Contains: String sizeID, Boolean gluten(true = gluten, false = glutenFree), List<String> toppingIDs, int MAX_TOPPING = 4	

ApplySpecialApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/apply_special**	
Use case: Desire to apply current specials to an existing cart of items	
API Operations	**Description**
POST/cart/{storeId}/{cartId}/special	Update the price of the cart depending on the special being applied

PurchaseApi **https://pizza-paradise.herokuapp.com/swagger-ui.html#/purchase**	
Use case: Desire to make purchase an order	
API Operations	**Description**
GET/cart/{storeId}/{cartId}/purchase	Get the receipt after successful purchase an order

PizzaCountApi https://pizza-paradise.herokuapp.com/swagger-ui.html#/pizza_count	
Use case: A customer is ordering pizza for the first time or has a group size that s/he is unfamiliar with and needs assistance deciding the amount to order	
API Operations	**Description**
GET/pizzaCount	Suggests the number of pizzas required to feed a given number of people

5.4 HttpStatusCode description

HttpStatusCode		
HttpStatus Code	**Code number**	**Description**
HttpStatus.OK	200	● When something is successfully found (example) ● GET/store/{storeId}/cart/{cartId} -> SUCCESS ● GET/store/{storeId}/cart/{cartId}/price -> SUCCESS
HttpStatus.CREATED	201	● When something is successfully created (example) ● POST/store/{storeId}/cart/{cartId}/add/pizza -> SUCCESS ● POST/store/{storeId}/cart/{cartId}/add/side -> SUCCESS
HttpStatus.NO_CONTENT	204	● When something is successfully removed (example) ● DELETE/store/{storeId}/cart/{cartId}/delete -> SUCCESS ● DELETE/store/{storeId}/cart/{cartId}/delete/side -> SUCCESS ● DELETE/store/{storeId}/cart/{cartId}/delete/pizza ->SUCCESS
HttpStatus.NOT_FOUND	404	● If the storeId is not found. ● If the pizzaSizeId is not found. ● If the store's gluten setting does not match with given gluten. ● If storeId and cartId are not matching.
HttpStatus.BAD_REQUEST	400	● If input given by user is bad. ● If given card expiration month is less than 1 or greater

		than 12.
HttpStatus.FORBIDDEN	403	● If adding new id when id already exist in the database. ● Trying to add specialItem to database and id already exist in the database

6. Appendices

6.1 Setup

1. MongoDB setup and configuration:
    - Visit MongoDB. It is available on: https://cloud.mongodb.com/user#/atlas/login
    - Create or login username and password
    - Click on COLLECTIONS
    - Click on CREATE DATABASE to create a new database, or
    - Click on existing database to view or edit collection within existing databases
2. Heroku setup and configuration:
    - Install Heroku CLI. It is available on: https://devcenter.heroku.com/articles/heroku-cli#download-and-install
    - To verify your CLI installation, use the **heroku --version** command
    - Stay in the CLI to enter your credentials, you may run **heroku login -i**
    - To create your first Heroku app run **heroku create**

6.2 Run application

1. Running Application
    - Clone and save it to your local machine
    - Open using terminal
        i. cd pizza-paradise
        ii. mvn spring-boot:run

6.3 Test application

1. Test Application
    - MongoDB must be running locally without a password
    - Open terminal
        i. mvn verify (to run automated tests)
        ii. mvn jacoco:report (to get detailed coverage reports)
            - Report found in target > site > jacoco > jacoco-resources > index.html (right click and open in browser)

6.4 Work timeline

This project's implementation was broken down into two-week sprints.

1. **"Sprint 1" due Friday (2019-10-11)run**
    - Change Pizza Store Name
    - API research and studies
    - MongoDB studies/overview
2. **"Sprint 2" due Friday (2019-10-25)**
    - Pizza Size API
    - Side API
    - Store API (carried over from Sprint 1 to Sprint 2)
    - Special API (carried over from Sprint 1 to Sprint 2)
    - Topping API (carried over from Sprint 1 to Sprint 2)
3. **"Sprint 3" due Friday (2019-11-08)**
    - Cart API
    - Appy Special API

    - Pizza Object
    - Suggested customer pizza count API
    - Testing APIs
4. **“Sprint 4” due Wednesday (2019-12-04)**
    - Refined APIs and testing
    - Purchase API
    - Design Documents
    - Presentation