TED ÜNİVERSİTESİ

CMPE 492

Senior Design Project II

Low-Level Design Report

Project Members:

Selin Akyurt - 17878041578

Melisa Uzulu - 3906408574

Recep Berkay Engin - 22271682594

Supervisor: Yücel ÇİMTAY

Jury Members: Aslı Gençtav, Venera Adanova

Table of Contents

1. Introduction

Cakery is here to help you with the problem of visiting bakeries and buy/order cakes for your celebrations and it will save your precious time making bakeries available for you with just few clicks.

Ordering a cake for a special occasion from a bakery can be time consuming and stressful task, especially with our busy schedules. Customized cakes, such as those for birthdays and engagement/wedding parties, require a lengthy process of visiting multiple bakeries, discussing preferences, comparing prices and more. That's where our project application "Cakery" comes in.

With Caker, you can easily access a list of local bakeries, view their products with detailed descriptions and place orders for customized or ready to eat cakes, all with few clicks with the comfort of your phone. Bakeries will then deliver the order to your desired location on the desired date and time. This way we will eliminate the time-consuming trips and consultations for you.

Cakery is also a platform where small businesses can showcase their unique products and reach a wider audience. With our app Cakery, we hope to give them the recognition they deserve for their hard work, while making the order process easier and more efficient for everyone involved.

The project is consisted of 3 different systems that requires careful planning and integration. Throughout the semester we are aiming to successfully put together a seller app, user app and an admin web portal which forms the Cakery using Flutter as a toolkit and Dart as a programming language. After successfully create our system basis, we will continue with testing the prototypes and the whole system together, add any necessary adjustments and improve the system to make the user experience easier. For the last step, we are hoping to extend the Cakery App for commercial use.

In this report, we provide information about the Low-Level design of the Cakery App, such as introduction, packages, class interfaces, glossary and references.

1.1 Object design trade-offs

For the object design trade-off part, we will describe and compare the usability, portability, reliability, supportability, performance, and security.

USABILITY VS PORTABILITY

Usability: Cakery App targets variable users. This is why we are using a user-friendly interface that does not require high learning curve. Overall system should be easy to use and understand without needing extensive guidance or instructions.

Portability: We are aiming to make sure our application can be easily used in different platforms, environments, and mobile devices.

- For the Cakery app, we believe usability is more important than portability, especially in early days of launching and developing our application. When we make sure Cakery app meets all the usability functions, we can extend the platforms, environments and devices to make sure our application is portable and available for everyone.

RELIABILITY VS SUPPORTABILITY

Reliability: Cakery should operate continuously and reliably without downtime to have a smooth ordering experience. To achieve this testing and monitoring could be handled very importantly and every update that application has should be tested before release.

Supportability: To make the workflow of the system smoother, occurred errors and bugs should be fixed with quick and easy updates and supported over time. Any error or fault happened in the payment section should be fixed immediately and users can contact the customer service for their payment problems.

- For the Cakery app, we believe reliability is more important over supportability. In early days of launch, we believe some errors and bugs are tolerable unless they cause major problems in the system. With systems updates we believe bugs and errors could be fixed with time.

PERFORMANCE VS SECURITY

Performance: Users of the Cakery app should be able to purchase their desired wants and needs without waiting for a long period of time since one of the starting points of this application is to solve the time problem for ordering cakes in bakeries. Human interactions, clicks/scrolls/selections should not take long to provide good user experience. Decent waiting time could be tolerable for the payment section.

Security: Security is one of the most important design goals for the Cakery. We aim to ensure the integrity and availability of user data and prevent unauthorized access and modification. Since we are offering buying and selling products in the application, we need to provide a secure payment system for users.

- For the Cakery app, we believe security is more important than performance since we need to provide a secure payment system in the application for our users. It is so important for our customers to feel save with their data using our app.

1.2 Interface documentation guidelines

Interface documentation guideline will show the format that has been used for the Class Interfaces in this Low Level Report. Class names, attributes and methods starts with lower

case letter. All attributes and methods are written in the camel case format such as ‘variableName’ and ‘methodName():’

Class Name
Description of Class
Attributes
Type of Attribute - Name of Attribute: Description of the attribute
Methods
NameOfMethod(ParametersOfMethod): Description and short rundown of method

1.3 Engineering standards (e.g., UML and IEEE)

The Unified Modeling Language (UML) guidelines are used in this report to describe the Cakery class interfaces, scenarios, use cases, diagrams, subsystem compositions and hardware depictions. UML is used for designing and documenting software systems that allows developers to communicate ideas and design in a clearer way.

1.4 Definitions, acronyms, and abbreviations

UML: Unified Modeling Language.

Bakery: a place where bread and cakes are made or sold.

Cake: an item of sweet soft food made from a mixture of ingredients, baked and sometimes iced or decorated.

Client: User end of the application.

Customer: a person who buys goods or services from a shop or business

Item: Any object, products, or thing that can be owned, possessed or used. Cake in this cake

Order: request (something) to be made, supplied, or served.

Product: an article or substance that is manufactured or refined for sale.

Seller: a person who sells something. Baker in this case

Small Business: privately owned corporation, partnership, or sole proprietorship that has fewer employees and less annual revenue than a corporation or regular-sized business.

2.Packages

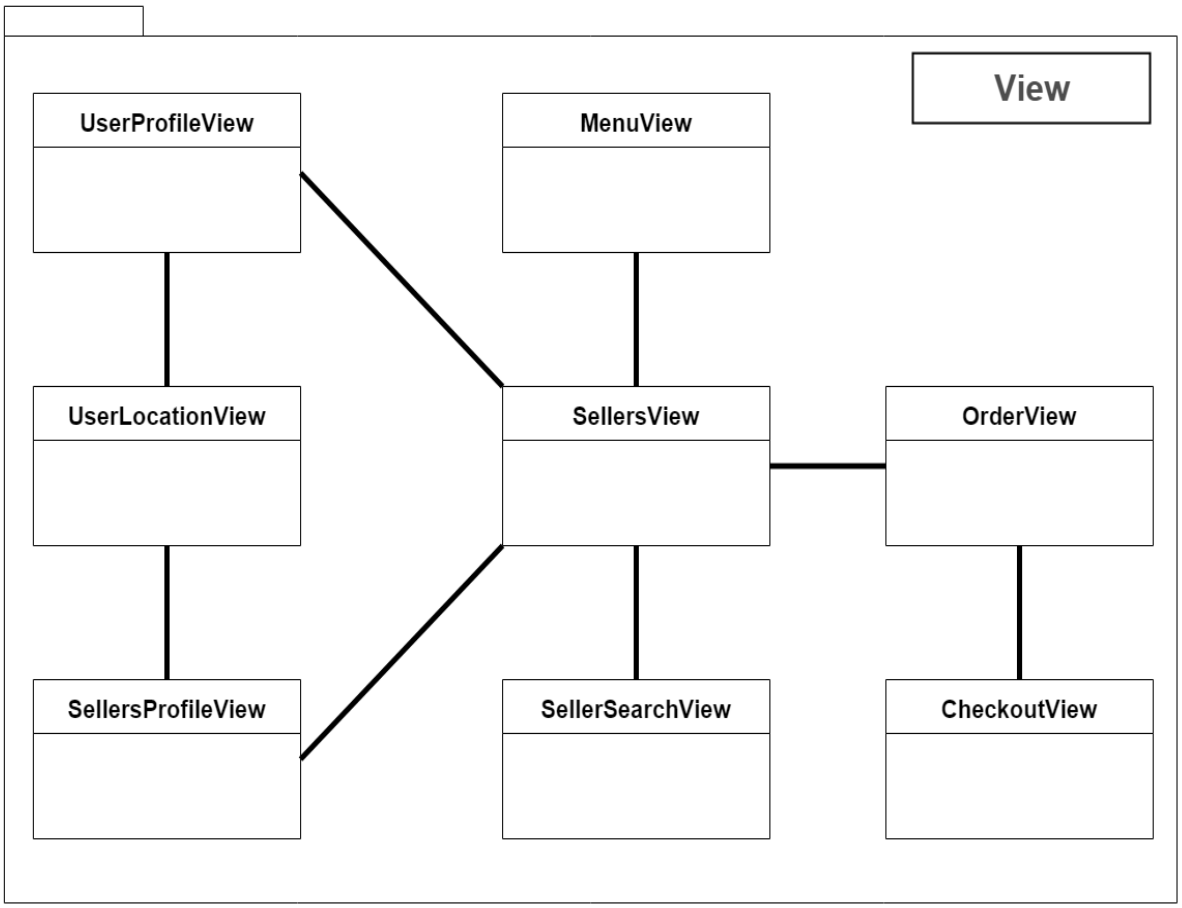
Overview

At the highest level, Cakery App consists of two parts, Client and Server. The server has three components; Request Handler, Data Layer, and Search Tools. The client also has three components; View, Presenter, and Model. In this part of the report, both the client and server sides of our application are examined in general. It shows how classes are divided into different packages. Individual descriptions for classes are specified in Class Interfaces.

Client

The client is the user interface layer where users and sellers interact with the system. Only users with Android devices can view this section. We have used the Model-View-Presenter pattern in our application: Inputs are handled by the View. The View knows and communicates with the Server that communicates with the View via an interface. Unlike MVC, there is a one-to-one connection between View and Presenter. The presenter manipulates the Model. The Model notifies the Server of changes within itself.

View



UserProfileView: This view class will display the User details and will be common to each user. However, the details will be displayed differently for different types of users. For example, the order page of users registered in the system will be different for each user. Each user's e-mail addresses will be different. In this section, the user will be able to view their orders and location.

SellersProfileView: This view class is the part where sellers can edit their menu and add/delete products from their menu. Each seller has its own page. The menus and products offered for sale by each seller may be different.

UserLocaitonView: This view class allows Users and vendors to display location details and change location information.

SellersView: This class allows users to see the active bakers' names and pictures of the bakery using the system.

MenuView: This class will represent detailed information about a seller's menus and items they have so that a user can get a general idea of that seller. Users can view the seller's most special cakes and classic cakes here.

CheckoutView: This view class will show the user the information of the products to be purchased. It will display the price information of the product or products to be purchased at

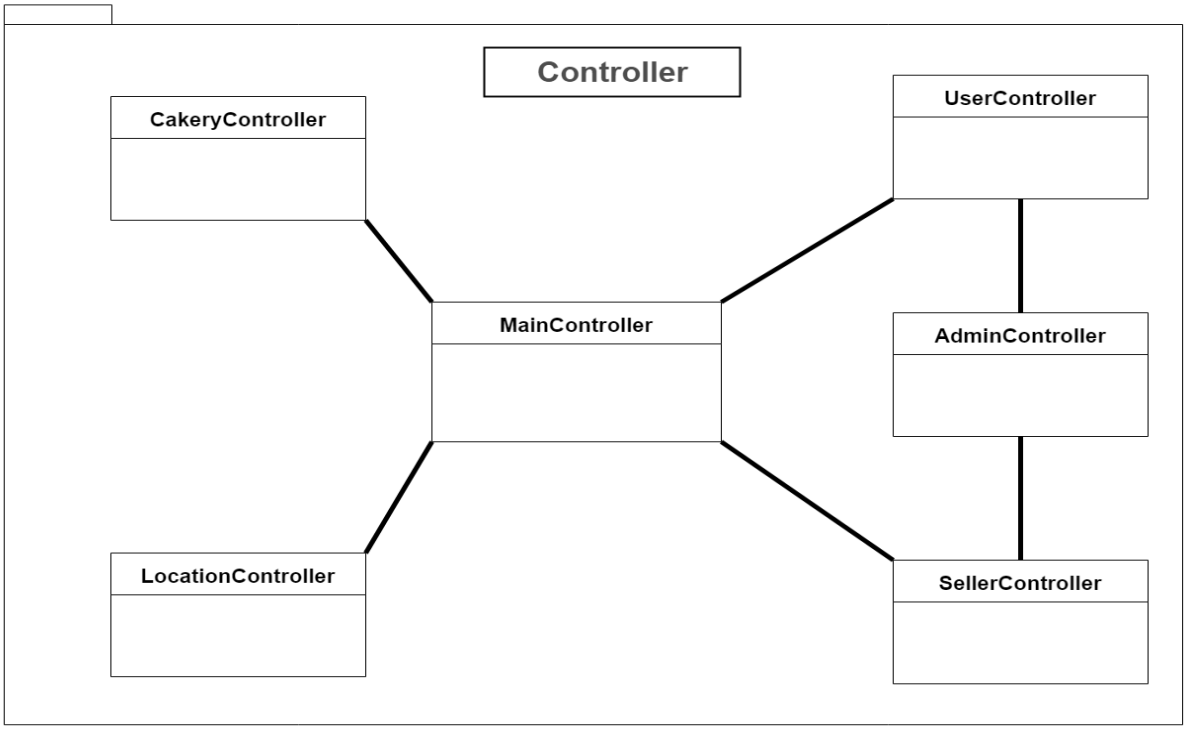
that moment, the address information to be delivered and many information about the order (payment methods, order number, etc.).

OrderView: This view class allows both the seller and the user to display the details of the order made by the user on their own pages.

SellerSearchView: This view class will be used to display vendor search results.

Server

Controller



MainController: This class is responsible for managing the inputs from the user and the vendor and collecting the necessary information to be displayed to the user/seller. This class gives the user; It uses other classes of controllers to communicate information about products, menus, and seller accounts. This class is also used to convey to the seller information about their menu and products, their orders. This class is the basic element of the system to ensure the flow of information between the user/seller and the system.

UserController: This controller class is responsible for performing user specific operations such as registering and logging in.

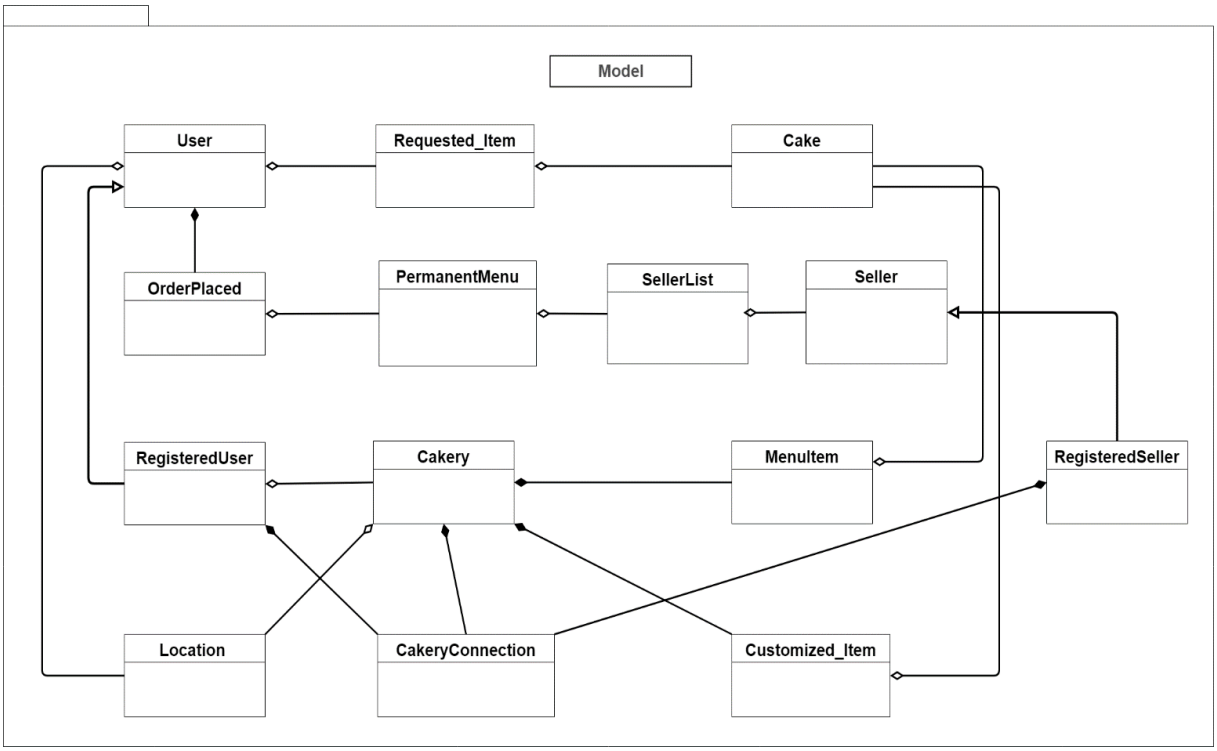
SellerController: This controller class is responsible for performing seller-specific operations such as registering and logging in.

CakeryController: This class is responsible for creating new menus for sellers, preparing the information of the products they will present to the user, and confirming the orders of the users.

LocationController: This class allows to communicate with the Google Maps API to determine the location of Users and vendors and to solve location related problems.

AdminController: This class is responsible for viewing the users and sellers information by the admin and for blocking/activating the user/seller by the admin.

Model



User: This class is the base class for all user types. It has basic features such as password, e-mail address, phone number, profile photo, location, order information. In addition, the User has the ability to make his/her own customizable cake. The user can follow the menus prepared by the seller.

Seller: This class is the base class for all vendor types. It has basic features such as password, email address, phone number, profile photo for the store, location, total earnings. Also, sellers have the opportunity to create their own menu. Vendors can show users prepared menus.

OrderPlaced: This class represents the record of the user who placed an order from a seller. It stores details of order information to provide users with data such as their previous orders.

Requested_Item: This class is basically a different representation of an ordered cake instance. It has a date, an address, and a user associated with it. It is the most fundamental asset of the system. When a user orders a cake, an instance of this class is initialized and passed to the

vendor. Then, users can view the order details of the requested order items in the next class, OrderPlaced.

PermanentMenu: This class inherits the Menu class. Vendors can create permanent menus where they can serve and sell on an ongoing basis. In addition, a permanent menu provides the user with a list of allowed cake types, both to guide customers on what to expect and to restrict the types of cakes customers want.

SellerList: This class represents the bakery type and Cakery will be compatible with these types. This class is used to identify the types of cakes and bakers.

Cake: This class represents items as an entity in the entire system. It has basic features (cake name, cake photo, cake features, cake price, etc.) and functions that make the system work and the items appear. An instance of this class will be created whenever a cake is searched for, added to the system, or requested.

MenuItem: This class represents the Seller's item list. The desired items will be added to the instance of this class and the items to be added to the menu will be selected from this section.

Customized_Item: This class represents the cake custom-personalized by a user. An instance of this class is initialized when a vendor's customizable portion is selected in the menu list. This class can see/change the ingredients and quantities in that cake when a user views this part.

CakeryConnection: This class stores Cakery connection related data.

Location: This class represents the physical location of Seller and Users. This class will be used to determine where Users are. Also, this class will be compatible with the Google Maps API as we will be using it as the main location data source.

ReqisteredUser: This class inherits the User class. Users who register via email will become a registered user. Registered users will be able to place orders and check their orders. To allow this functionality, RegisteredUser stores user information such as email, profile picture, and information such as current locations.

ReqisteredSeller: This class inherits the Vendor class. Merchants who sign up via email will be an active registered seller. Registered sellers will be able to sell and display their products to users and control their orders/earnings. To allow this functionality, RegisteredSeller stores seller information such as email, profile picture, and information such as current locations.

Cakery: This class represents vendors. Cakery is at the point where people can check whether they are in the necessary proximity to the oven in question and get information about the oven. Cakery helps both commercial place owners and individual users to create such a place to benefit each other for their own purposes. While users aim to have the cakes they want, sellers aim to deliver the cakes they create to people. This is thanks to Cakery. seller needs a name, a location represented by the Location class, and a list of menus represented by the SellerMenu class and shared with users. This class also stores order data to provide functions that Cakery App offers.

3.Class Interfaces

Cakery, consists of two sub-application and a web portal. In this section, class descriptions, attributes and methods for each system is explained briefly.

Class Name: login.dart
Description of Class: This class aims to create and operate login processs.
Attributes
GlobalKey <FormState>formKey: uniquely identifies the form and allows validation in the form fields TextEditingController emailController: holds the current value of email field TextEditingController passwordController: holds the current value of the password field in hash
Methods
dynamic formValidation(): validates controllers dynamic loginNow() : checks if credentials are authenticated and performs login Future <dynamic> readDataAndSetLocally(): saves the data locally for the key Widget build(): creates the login page UI

****login.dart is a class which is also visible in User App of Cakery.**

Class Name: register.dart
Description of Class: This class aims to create and operate register process.
Attributes
GlobalKey <FormState> formKey: uniquely identifies the form and allows validation in the form fields TextEditingController nameController: holds the current value of name field TextEditingController emailController: holds the current value of email field TextEditingController passwordController: holds the current value of password field TextEditingController confirmPasswordController: holds the current value of password field TextEditingController phoneController: holds the current value of phone field TextEditingController locationController: holds the current value of location field

XFile imageXFile: file format to store picture Position position: holds information of position List <Placemark> placeMarks: holds information about location String sellerImageUrl: holds URL for seller image LocationPermission? permission: holds permission for enabling device location informations String completeAdress: holds complete address
Methods
Future <void> getImage(): gets image from gallery Future <Position> getCurrentLocation(): checks permission and gets current position Future <void> formValidation(): validating form creditentials void authenticateSellerAndSignUp(): creates user account using Firebase Authentication, saves user information to Firestore using saveDataToFirestore() Future <dynamic> saveDataToFirestore(): saves seller info to Firestore collection and saves data locally Widget build(): creates the register page UI

****register.dart is a class which is also visible in User App of Cakery with few additional information but the logic holds the same.**

Class Name: menu_upload_screen.dart
Description of Class: This class aims to upload picture and get menu information.
Attributes
XFile imageXFile: file format to store picture ImagePicker picker: instance of a plugin for picking image from library and taking picture with camera TextEditingController shortInfoController: holds the current value of menu info TextEditingController titleController: holds the current value of menu title bool uploading: state of uploading process String uniqueIdName: unique ID of menu
Methods
dynamic defaultScreen(): creates the menu upload page UI dynamic takeImage(): creates dialogs for image uploading options dynamic CaptureImageWithCamera(): captures image using the device camera and updates UI

dynamic pickImageFromGallery(): picks image from users gallery and updates the UI dynamic menusUploadFormScreen(): represents the screen that allow user to upload a new menu to account dynamic clearUploadFormScreen(): clears the state of the form dynamic validateUploadForm(): validating form creditentials dynamic saveInfo(): saving menu information to database dynamic uploadImage(): upload an image file to Firebase and return download URL of the uploaded image Widget build(): to conditionally render based on image file state

Class Name: items.dart
Description of Class: This class is generated for creating a template for displaying the Json response from Firestore.
Attributes
String menuID String sellerUID String itemID String title String shortInfo String thumbnailUrl String status String longDescription int price Timestamp publishDate: represents Coordinated Universal Time standard, a point in time
Methods
Map <String, dynamic> toJson(): converts instance to a JSON object

****We use model structure in Cakery's logic design. Just like items.dart, we have address.dart and menus.dart models, they have different attributes and standardization but logic remains the same. They're used for creating a template for displaying the Json response from Firestore throughout the classes.**

Class Name: info_design.dart
Description of Class: This class is the design of menu information. Creates the UI context for the model created for displaying a menu.
Attributes
Menus model: instance of menu model BuildContext Context: to return the widgets that is built using this context
Methods
Widget build(): builds the UI for displaying address. deleteMenu(): functionality for the delete button in a menu card, connects to Firestore and deletes corresponding menuID.

Class Name: order_details_screen.dart
Description of Class: This class creates order information screen and retrieves desired data from Firebase.
Attributes
String orderStatus: holds status of order-normal or ended String orderByUser: holds user ID of customer String sellerId
Methods
dynamic getOrderInfo(): gets information of the current order void initState(): updates state of order info Widget build(): creates order details screen UI and retrieve data from Firestore

Class Name: order_card.dart
Description of Class: This class is a helper widget class which aims to standardize an order card.
Attributes
Int itemCount String orderID List<String> seperateQuantitiesList: keeps a list of quantity
Methods
Widget build(): returns placedOrderDesignWidget() helper method

Class Name: earnings_screen.dart
Description of Class: This class aims to process earnings screen UI and logic.
Attributes
double sellerTotalEarnings
Methods
dynamic retrieveSellerEarnings(): void initState(): updates state of earnings info Widget build(): earnings screen UI

****Classes below are for the Cakery User App**

Class Name: order_card.dart
Description of Class: This class represents order card UI and keeps desired information of the order card for further use.
Attributes
int itemCount: holds number of item List <DocumentSnapshot> Data: holds instances of DocumentSnapshot instances String orderID List <String> seperateQuantitiesList: holds separated quantities.
Methods
Widget build(): creates a List Tile for displaying ordered items Widget placedOrderDesignWidget(): called by build, cerates a UI for ordered items

Class Name: addres_design.dart
Description of Class: This class is the design of address information. Creates the UI context for the model created for address.
Attributes
Adress model: instance of address model Int value String addressID double totalAmount String sellerUID
Methods
Widget build(): builds the UI for displaying address.

Class Name: search_screen.dart
Description of Class: This class aims to create a search bar for restaurant names, creates UI and performs data retrieval.
Attributes
Future <QuerySnapshot> restaurantDocumentsList : holds list of restaurants greater than or equal to the text entered String sellerNameText
Methods
Widget build(): creates UI of search screen, updates state and calls initSearchingRestaurant() method for use initSearchingRestaurant(): calls restaurantDocumentsList and gets a Firestore instance of list

Class Name: placed_order_screen.dart
Description of Class: This class aims to show placed order information to user and seller.
Attributes
String addressID double totalAmount String sellerUID String orderID
Methods
addOrderDetails(): writes order details for user, writes order details for seller, updates state and clears cart when complete Widget build(): creates a UI for placed order screen

Class Name: sellers.dart
Description of Class: This class is generated for creating a template for displaying the Json response from Firestore.
Attributes
String sellerName String sellerUID String sellerAvatarUrl String sellerEmail
Methods
Map <String, dynamic> toJson(): converts instance to a JSON object

Class Name: sellers_design.dart
Description of Class: This class is the design of seller information. Creates the UI context for the model created for sellers' representation.
Attributes
Sellers model: instance of seller model BuildContext Context: to return the widgets that is built using this context
Methods
Widget build(): builds the UI for displaying sellers.

Class Name: cart_screen_state.dart
Description of Class: This class aims to create UI and perform data retrieval.
Attributes
String sellerUID List <int> separateItemQquantityList num totalAmount: holds the total amount for the cart
Methods
Widget build(): performs clearCartNow() method for related buttons, performs query for displaying items from Firestore, calculates and displays total amount and calls cart_item_design.dart class for UI of cart screen

Class Name: menus_screen.dart
Description of Class: This class represents the menu screen of a bakery.
Attributes
Sellers model: instance of seller model
Methods
Widget build():builds the UI for menu screen and gets Firestore instance of menus of the active seller

Class Name: items_screen.dart
Description of Class: This class represents the item screen of a menu.
Attributes
Menus model: instance of menus model
Methods
Widget build(): calls items_design() for items UI and gets Firestore instance of the corresponding menu

Class Name: cart_item_counter.dart
Description of Class: This class is for creating an assistant method that will be used to keep track of number of cart items, called by cart_screen.
Attributes
cartListItemCounter: holds the count of items in the user's cart. int count: getter method that returns the value of cartListItemCounter
Methods
dynamic displayCartItemsNumber(): responsible for updating the value of cartListItemCounter and notify listeners that the value has changed

Class Name: total_amount.dart
Description of Class: This class is for creating an assistant method that holds the current total amount of items in the user's cart and managing the state information accordingly.
Attributes
double totalAmount
Methods
displayTotalAmount(): responsible for updating the value of totalAmount and notifying listeners that the value has changed

Class Name: history_screen.dart
Description of Class: This class displays a list of orders made by the user in descending order of their order time.
Attributes
There are no attributes in this class but inherits attributes and methods from other classes.
Methods
Widget build(): responsible for displaying the UI of the screen and the list of order cards respectively.

**These classes can be modified based on future design and logic decisions.

4. Glossary

Dart: Dart is a programming language developed by Google. It's an object-oriented language that can be used for a variety of applications including web and mobile development.

Flutter: Flutter is an open-source mobile application development framework created by Google. It uses the Dart programming language and provides a rich set of pre-built widgets and tools to help developers create high performance, visually attractive mobile applications for both Android and IOS platforms.

Firestore: Firestore is a cloud based NoSQl document database service offered by Google. It provides real-time data synchronization and querying capabilities, making it a popular choice for building mobile and web applications that require fast, reliable data storage and retrieval.

Json: JSON stands for "JavaScript Object Notation," and is a light ware data interchange format that is easy for human to read and write, and easy for machines to pase and generate. It is often used in web applications to transmit and store data in a structured format and can be used easily converted to and form other data formats like XML and CSV.

UI: UI stands for "user interface," and refers to the graphical layout and design of a software application or website. A good UI design helps users interact with the application more efficiently and effectively, by providing clear and intuitive navigation, visual cues, and feedback.

UML: UML stands for "Unified Modeling Language," and is a standardized notation for creating diagrams and models that represents various aspects of a software systems. It is often used in software development to describe the architecture, behaviour, and relationships between different components of a system.

5. References:

[1] "ACM Code of Ethics and Professional Conduct"

[2] "Object-Oriented Software Engineering, Using UML, Patterns, and Java," 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13- 047110-0.

[3] "UML 2 Class Diagramming Guidelines." Agile Design, agilemodeling.com/style/classDiagram.htm.

[4] "Nearby Places for Android | ArcGIS for Developers." System Requirements | ArcGIS API for JavaScript 4.8, https://developers.arcgis.com/exampleapps/nearby-android/.

[5] IBM, "UML - Basics," June 2003. [Online]. Available: http://www.ibm.com/developerworks/rational/library/769.html

[6] "Cakery High-Level Design Report"