

Florida International University
Knight Foundation School of Computing and Information Sciences

Software Engineering Focus

Final Deliverable

Project Title: Project Manager

Team Members:

Jose Hernandez, Jose Pujol, Naor Vidal

Product Owner(s):

Aydin Danaci

Mentor(s):

Dr. Masoud Sadjadi

Instructor: Dr. Masoud Sadjadi

The MIT License (MIT)
Copyright (c) 2016 Florida International University

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Abstract

This document presents the information necessary to gain a good understanding of Project Manager, a cross-platform mobile application that allows employees of Ukay Nationwide Furniture Delivery Service to better track their drivers, and the status of the deliveries as the drivers complete their daily assigned routes.

Table of Contents

Introduction.....	5
Current System.....	5
Purpose of New System.....	5
User Stories.....	6
Implemented User Stories.....	6
Pending User Stories.....	6
Project Plan.....	8
Hardware and Software Resources.....	8
Sprints Plan.....	9
System Design.....	9
Architectural Patterns.....	9
System and Decomposition.....	10
Design Patterns.....	11
System Validation.....	11
Glossary.....	11
Appendix.....	13
Appendix A - UML Diagrams.....	13
Appendix B - User Interface Design.....	15
Appendix C - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents.....	16
References.....	36

INTRODUCTION

Current System

Ukay Furniture Delivery currently tracks all deliveries manually through the use of spreadsheets, and physical forms. There is no advanced-integrated system that allows for live checks on item deliveries, the status of the delivery, nor any checks to see if drivers are following a predetermined route. This introduces many variables that can cause delays in shipments, as well as the loss of confirmations of receipts, meaning that the administrative dispatch team may lose the ability to confirm if a delivery was ever attempted or completed for any one delivery.

Purpose of New System

The purpose of the new system, embodied by the Ukay Delivery Application, is to revolutionize the furniture delivery process by introducing a suite of digital tools aimed at enhancing efficiency, transparency, and customer satisfaction. The application serves as a centralized platform that seamlessly connects customers, dispatchers, drivers, and administrators. For customers, it offers real-time tracking of deliveries, flexible scheduling, and easy confirmation of receipt. Dispatchers benefit from dynamic delivery assignment capabilities, real-time monitoring, and direct communication channels with drivers. Drivers are empowered with optimized route planning and digital proof of delivery functionalities, while administrators gain access to robust data analytics, system configuration options, and comprehensive reporting tools. Overall, the application is designed to streamline operations, minimize manual errors, and foster a more connected and responsive delivery service.

USER STORIES

The following section provides the detailed user stories that were implemented in this iteration of Project Manager. These user stories served as the basis for the implementation of the project's features. This section also shows the user stories that are to be considered for future development.

Implemented User Stories

Driver User Stories

As a driver, I want to view a list of my assigned deliveries for the day, so that I can plan my schedule effectively.

As a driver, I want to receive optimized routes for my deliveries, so that I can reach my destinations using the quickest and safest routes.

As a driver, I want to view customer details and delivery instructions, so that I can deliver items according to the customer's specifications.

Pending User Stories

Administrator User Stories

As an administrator, I want to be able to create, update, and deactivate user accounts, so that I can manage access levels for dispatchers, drivers, and customers efficiently.

As an administrator, I want to access a dashboard where I can view analytics on delivery times, customer feedback, and driver performance, to help in decision-making and strategy development.

As an administrator, I want to configure application settings, including notification preferences and operational parameters, to ensure the system runs according to our organizational needs.

As an administrator, I want to generate comprehensive reports on delivery efficiency and customer satisfaction, to provide insights into business performance and areas for improvement.

As an administrator, I need to oversee the maintenance and updates of the app, ensuring that the system is up-to-date, secure, and running smoothly, to avoid any operational disruptions.

Dispatcher User Stories

As a dispatcher, I want to assign and reassign deliveries to drivers based on their current location and availability, to optimize delivery routes and times.

As a dispatcher, I need to monitor the real-time status and location of all deliveries, to ensure timely deliveries and to quickly address any issues that arise.

As a dispatcher, I want to communicate directly with drivers to provide updates or assistance, to maintain a seamless delivery process and respond promptly to any challenges.

As a dispatcher, I want to be able to quickly address and resolve any customer queries regarding delivery status or concerns, to maintain high customer satisfaction levels.

Customer User Stories

As a customer, I want to track the real-time status of my furniture delivery, so that I can be informed about its progress and plan my schedule accordingly.

As a customer, I want to schedule my delivery at a time convenient for me, ensuring that I am available to receive my furniture without any hassle.

As a customer, I want to confirm receipt of my items using a digital signature, to provide proof of delivery and to finalize the transaction.

As a customer, I want to easily submit feedback about my delivery experience through the app, to help the company improve their service and address any issues I may have faced.

PROJECT PLAN

An intensive phase of research was conducted throughout the span of two sprints that allowed us to gather all of the user's requirements, as well as going through multiple iterations of the system design phase. Throughout this process, the methodologies of how data is stored and processed underwent several revisions before converging to our implemented solutions.

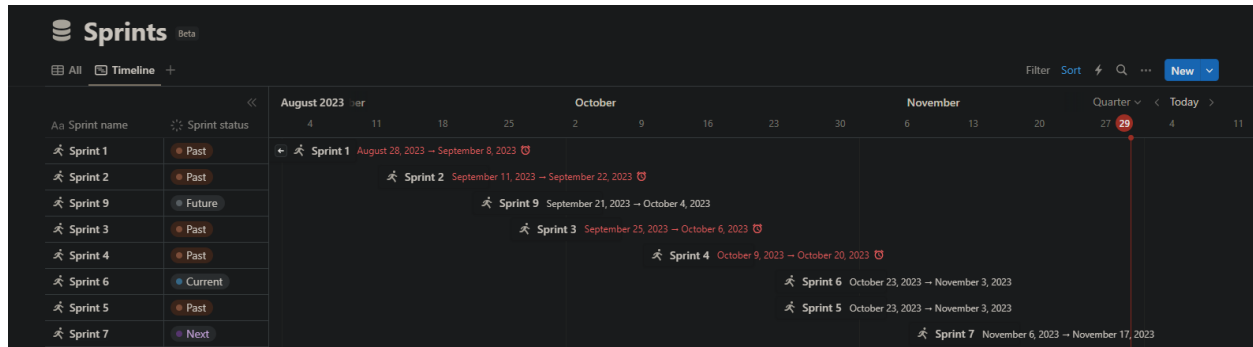
Because of the time spent developing the initial system, as well as the research of the technologies, we were unable to fully deploy the app. However, it is in a near-deployment ready state. More research and a cost-analysis sheet needs to be constructed, as the client requests this service to run on the cloud.

Hardware and Software Resources

The following is a list of all hardware and software resources that were used in this project:

- Hardware:
 - Samsung Galaxy S21+,
 - iPhone 11 Pro Max
 - M1 MacBook Air
 - Windows Desktop Computer
- Software:
 - Android Studio
 - Docker
 - Visual Studio Code
 - PostgresDB
 - Flutter
 - Dart
 - Python
 - PyEnv
 - FastAPI
 - Ngrok
 - Xcode

Sprints Plan



SYSTEM DESIGN

This section contains information on the design decisions that went into this project. The architecture patterns are outlined and explained. The entire system is shown in a package diagram and the subsystems are explained. Finally, the design patterns used in the project are discussed.

Architectural Patterns

Multiple architectural patterns were utilized in order to make this project successful. Some include:

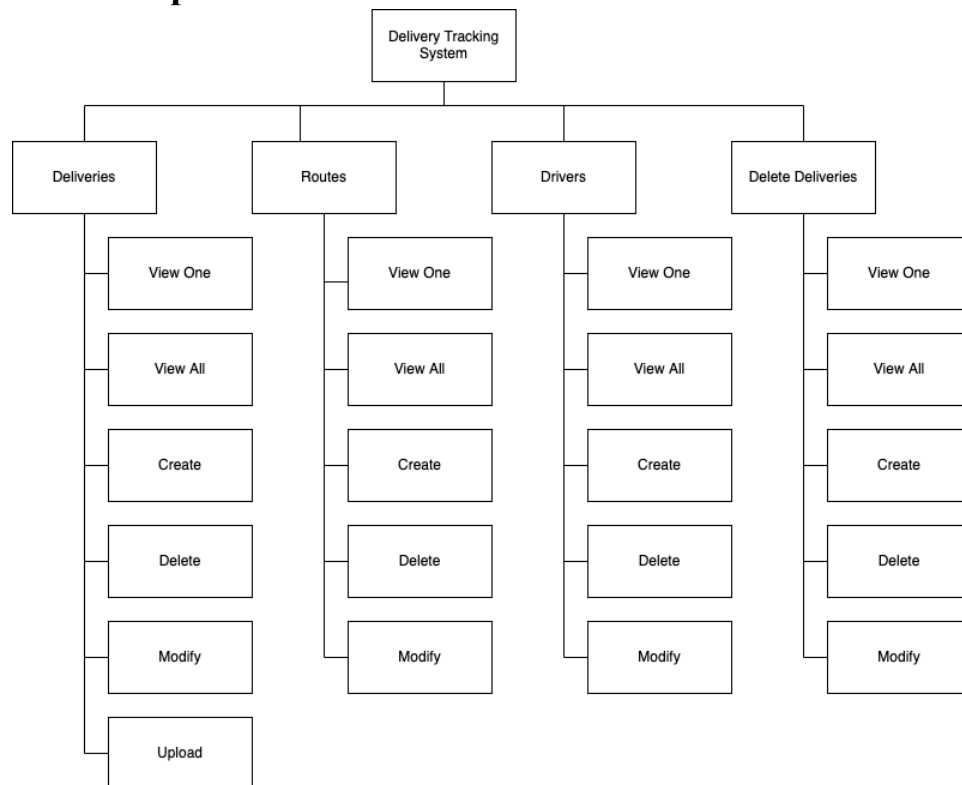
- Client-Server Model,
- Model, Viewer, Controller Model,
- And Microservices Model.

Each of these architectural models assisted us in breaking down the entirety of the system into its key components. The client-server model allowed us to proficiently build a server that supports multiple connections from multiple clients at any given moment. Additionally, when deployed, a load-balancer can be implemented such that the server does not get overloaded with requests. Data sharding may be practiced as well in order to build a robust database.

The Model, View, Controller Model aids in the separation of responsibilities. The model represents data models which define how we store information of the drivers, their deliveries, and their routes. Viewers represent where the data is viewed. Being a cross-platform application, a viewer could be anything such as a mobile phone, or a web browser. Controllers represent the server itself, in which it controls the flow of data, as well as the processing of data.

The Microservices model aptly describes how this application should be deployed. Each core-service was built to be individually deployable such that each may be scaled independently. For example, if there are 10,000 clients connecting to the server, and it begins to return data with significant latency, a load balancer may be implemented, which is later backed by more instances of the server, allowing the concurrent connection of 10,000 clients. Additionally, multiple instances of the database may be deployed as well to remediate or avoid a similar issue.

System and Decomposition



Design Patterns

Multiple design patterns were implemented throughout the system, such as Data Access Objects (DAOs), Builder classes, and Singletons. Each of which ensures data validity, and quicker processing times. Additionally, the project closely aligns with *some* of the SOLID principles.

SYSTEM VALIDATION

The system validation process for the Ukay Delivery Application is a multi-faceted approach designed to ensure the application's functionality, reliability, and user experience meet the highest standards. It encompasses a series of rigorous testing phases, including unit testing to validate individual components, integration testing to ensure seamless interaction between different parts of the application, and system testing to evaluate the application's overall performance under various conditions. User acceptance testing is then conducted with real-world scenarios to gather feedback from actual users to ensure the system is intuitive and meets their needs. This process also includes validating the application against compliance and security standards to safeguard user data. Post-deployment, the application will undergo continuous monitoring and periodic audits to ensure ongoing performance and security, with feedback loops in place to facilitate regular updates and improvements based on user input and changing business requirements.

GLOSSARY

SOLID - Acronym for the first five object-oriented design (OOD) principles by Robert C. Martin.

Model Viewer Controller Model - a pattern in software design commonly used to implement user interfaces, data, and controlling logic.

Microservices Model - an architectural style for developing applications. Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility.

Client-Server Model - a model of interaction in which a program sends a request to another program and awaits a response. The requesting program is called a client; the answering program is called a server.

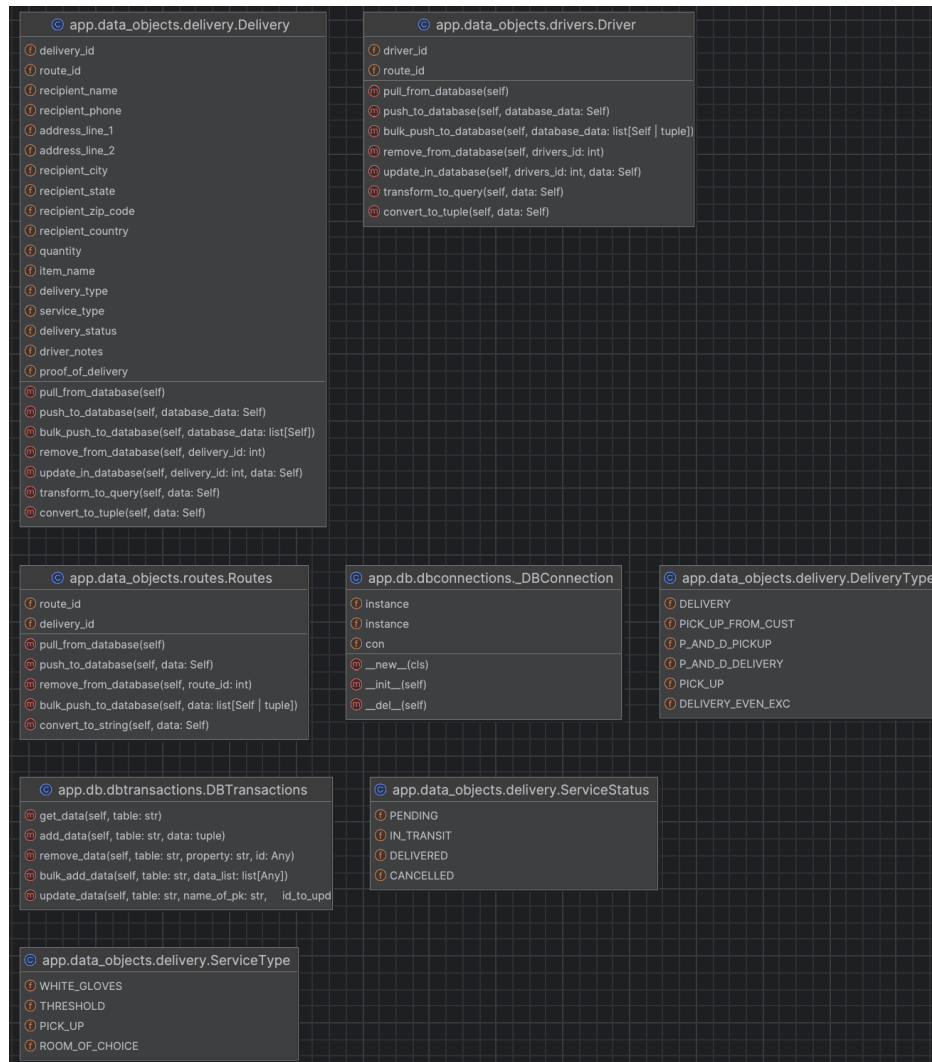
Data Access Object (DAO) - separates a data resource's client interface from its data access mechanisms. Adapts a specific data resource's access API to a generic client interface

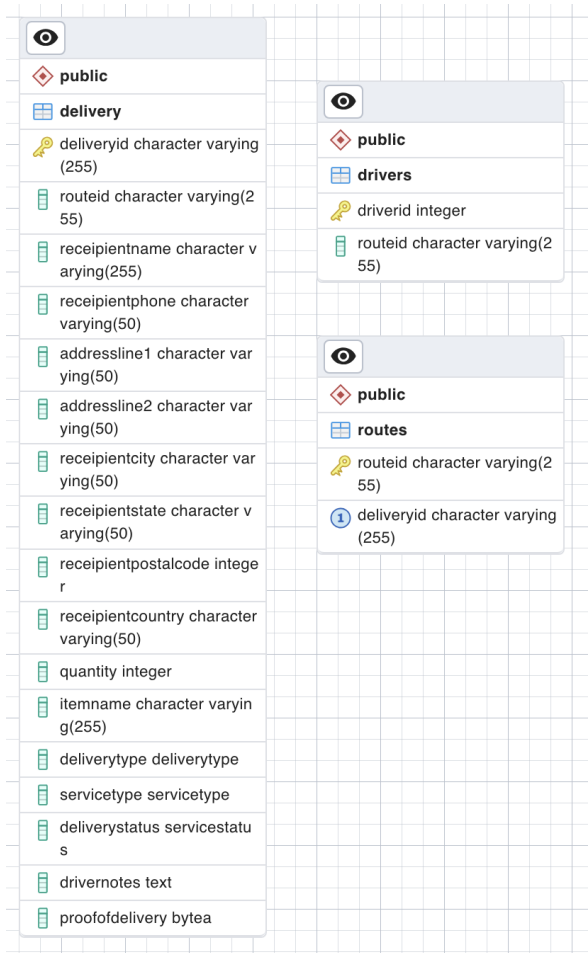
Singleton - Provides access to a shared resource using a single, shared class instance.

Builder Class - Separates the construction of a complex object from its representation so that the same construction process can create different representations.

APPENDIX

Appendix A - UML Diagrams



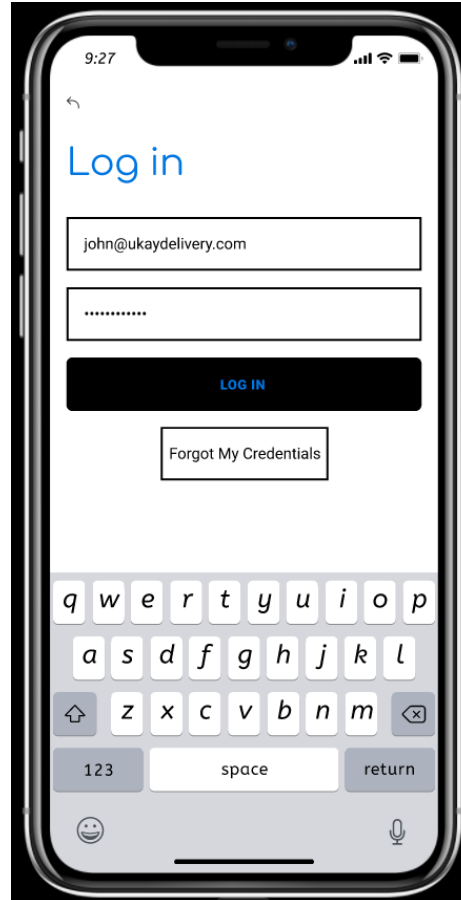


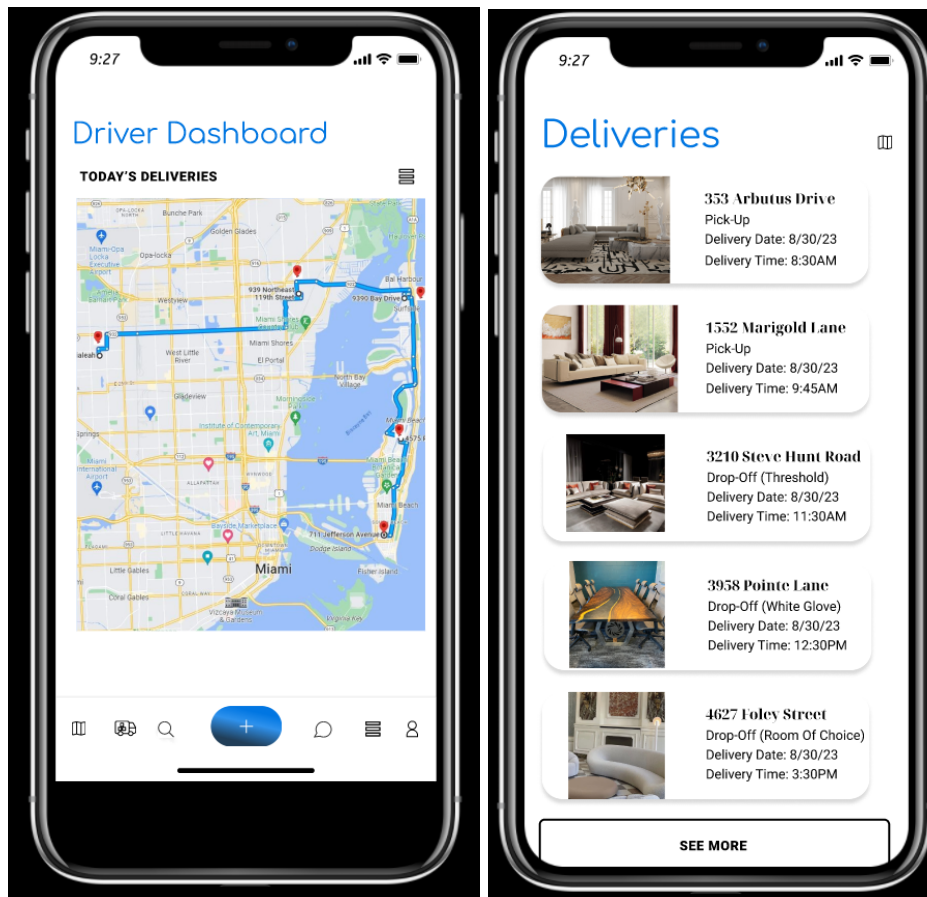
Appendix B - User Interface Design

Home Screen



Login Screen



*Driver Home Screen**Deliveries Screen*

Appendix C - User Manuals, Installation/Maintenance Document, Shortcomings/Wishlist Document and other documents

(Begins on the next page.)

User Manual for Ukay Mobile Delivery Application

User Manual for Furniture Delivery App

Introduction

Welcome to the Ukay Furniture Delivery App! This user-friendly app is designed to help you manage furniture deliveries efficiently. Whether you're tracking routes, managing inventory, or handling today's deliveries, this app has you covered.

Getting Started

Installation

1. **Download the App:** Download the Furniture Delivery App from your respective app store.
2. **Open the App:** Tap on the app icon to open it.
3. **Login/Signup:** If you're a first-time user, sign up with your details. If you already have an account, log in with your credentials.

Navigating the App

The app has three main sections accessible via the bottom navigation bar: Map, Inventory, and Today's Deliveries.

1. Map

- **Accessing the Map:** Tap on the 'Map' icon in the bottom navigation bar.
- **Viewing Routes:** Here, you can view the delivery routes. Zoom in/out to get a better view of the map.

2. Inventory

- **Accessing Inventory:** Tap on 'Inventory' in the bottom navigation bar.
- **Viewing Items:** A list of inventory items is displayed. Scroll through to view all items.
- **Details of an Item:** To view more details about an item, tap on 'Details' next to the item.

3. Today's Deliveries

- **Accessing Today's Deliveries:** Select 'Today's Deliveries' from the bottom navigation bar.
- **Viewing Deliveries:** A list of today's deliveries is shown.
- **Managing Delivery:** Tap on a delivery to view options for 'View', 'Modify', or 'Delete'. Choose an option to proceed:
 - **View:** Check the details of the delivery.
 - **Modify:** Make changes to the delivery details.
 - **Delete:** Remove the delivery from the list.

Uploading Photos

- **Accessing Upload Feature:** This feature is available in the 'Today's Deliveries' section.
- **Uploading a Photo:** Tap on the camera icon (Floating Action Button) to upload a photo.
 - **Select Image:** Choose an image from your gallery to upload.
 - **Upload Image:** After selecting the image, tap on 'Upload Image' to upload it to the server.

Tips for Efficient Use

- **Regularly Update Inventory:** Keep the inventory updated for accurate tracking.
- **Check the Map Routinely:** Stay updated with delivery routes to optimize delivery times.

- **Utilize the Photo Upload:** Use the photo upload feature to keep a visual record of deliveries.

Troubleshooting

- **Login Issues:** Ensure you're using correct login credentials. Reset your password if necessary.
- **Map Not Displaying:** Check your internet connection or try restarting the app.
- **Photo Upload Not Working:** Ensure you have given the app permission to access your gallery.

Support

If you encounter any issues or have questions, please contact our support team at info@ukaydelivery.com

This user manual is intended to guide you through the basic functionalities of the Furniture Delivery App. For more detailed information or specific queries, please refer to our in-app help section or contact support.

High Level Design and Implementation

Background

Ukay Furniture Delivery currently tracks all deliveries manually through the use of spreadsheets, and physical forms. There is no advanced-integrated system that allows for live checks on item deliveries, the status of the delivery, nor any checks to see if drivers are following a pre-determined route. This introduces many variables that can cause delays in shipments, as well as the loss of confirmations of receipts, meaning that the administrative dispatch team may lose the ability to confirm if a delivery was ever attempted or completed for any one delivery.

The purpose of this application is to provide a purpose-built solution that implements the pre-existing systems, and further enhances on it.

Solution

The client requested that we develop a mobile application that allows him, his dispatchers, and his drivers be able to upload a list of deliveries with pertinent details to this application. The application allows drivers to view a list of deliveries assigned to them, and view the client information such as the items to be delivered, client address, and contact information.

Once the driver verifies that the information is correct, they may pick to begin the delivery. The application will then open the appropriate navigation app on their phone, which will have the clients' addresses pre-populated, and navigation is ready to begin.

Drivers are able to then update the status of a delivery, upload a Proof of Delivery, delay a delivery, or perform other actions based on what is going on at their level.

Administrators at the office are able to view these updates live as the driver progresses through their routes.

If any delivery is delayed, the driver is able to initiate a text message conversation with the client to inform them of the status of their delivery, as well as provide an ETA.

Additionally, the app will automatically notify the client that the driver is en-route, and is approaching the provided address such that delays at the destination are minimized.

Technical Implementations

The mobile application (front-end) was designed such that it works across all major platforms (Android, iOS, and Web Applications). Data storage, data retrieval, and data processing happens independently from the application interface in order to ensure that we are able to provide maximum performance at the application level. This allows the solution to be scalable and robust such that it can be hosted using any computing platform, and that failures that happen on a per-device basis is isolated to the conflicting device, and will not affect other users.

The business logic for data storage, data retrieval, and data processing (back-end) was designed in such a way that data is received and sent using HTTP endpoints. This allows future users and developers of the solution to easily integrate the suite using any platform, as well as ease conflicts if any of the technology changes because of updates, or design choices. Additionally, using HTTP endpoints allows developers to test the business logic without the need of launching the mobile application, allowing development to happen in parallel, and independently of each other.

Technical Limitations

The business logic of the suite is currently unsecured, and is not ready for a production environment. Security is a primary concern as this makes the application susceptible to cyberattacks from many different vectors. Additionally, communications between the front-end and back-end are also unsecure.

The front-end of the application was implemented in a barebones fashion. It does not support data caching, transaction queuing, as well as other features that further improves its performance. If the back-end of the application is not running, the front-end is useless because there is no local cache of delivery information, routes, and photos/spreadsheets can not be uploaded to the back-end.

Technical Implementation

Technology

Front-End

Flutter

Google Maps Flutter Library

Back-End

Python 3.11

FastAPI

Pydantic

Psycpg2

Ruff

Pandas

Docker

Postgres

Front-End

The front-end utilizes Google's Material designs, their Maps Library, and connects to our backend using HTTP Endpoints. The entirety of the front-end is built using Flutter, which allows us to easily develop the application once, and run it across multiple platforms.

A typical Flutter project tree will contain directories for each platform, but all development happens in the 'lib' directory.

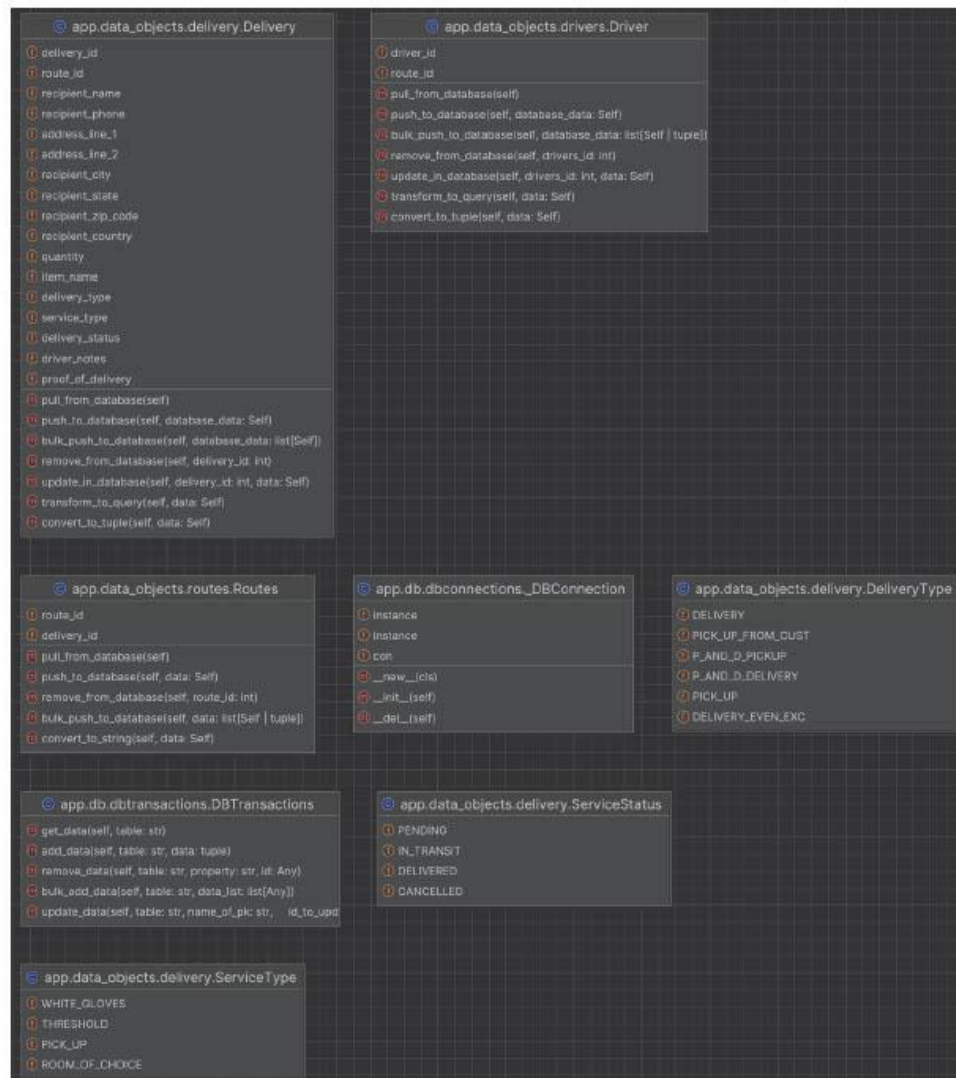
In our case, all "views" are in the pages directory, utility classes and functions are in the "utilities" directory.

Within the utilities directory are the api and dataObjects directories. the api directory contains utility functions to interact with the backend, the dataObjects directory will contain classes for a 1:1 data representation with the back-end.

Back-End

The backend is run independently from the front-end and supports connections to multiple clients through HTTP endpoints. The Python backend holds a temporary data-store that reflects the tables and their columns as stored in Postgres.

The backend works through DAOs as well as utilizing the builder paradigm to build data objects. Data validation is handled automatically through Pydantic. Once data is received and ready to be pushed to either the front-end or the database, the backend will convert it to either a JSON response, or a Postgres query. Data can be manipulated in the backend, but once it is committed, it must be pulled again, and the resultant must be modify then later committed.



The Python files in the `data_objects` module are 1:1 representations of their Postgres table counterparts. Each data object includes some utility functions that aids in sending, receiving, updating, and deleting data from the database such that no SQL queries are need to be directly written.

The `_DBConnection` class is a singleton that ensures only one Postgres connection is opened and handled by `Psycopg2` at a time. `DBTransactions` holds SQL query strings that are used to execute and commit changes from the data objects.

API

API Calls are defined using `FastAPI`, which will automatically generate an OpenAPI JSON document. Each data object has its own route with corresponding methods to Create, Read, Update, and Delete entries from/to the database. Whenever data is received to be uploaded to the database, it is received as JSON in the request body from the front-end. Once the data is received, it is converted to an object and data validation is performed synchronously using `Pydantic`. The API routes hold no business logic, other to ensure that data movement is correct and valid.

Excel files which hold delivery, route, and driver info are received as byte information, converted in-app as an Excel file, turned into a dataframe using `Pandas`, and finally converted to a data object. Once this process is complete, each row from the Excel file gets uploaded to the database.

The API does not return data objects from the POST or PUT HTTP requests. Once a request is received, and the data is validated, the API will return an HTTP 201 OK. This status code represents that the data was received and accepted. Because synchronous data uploads is not guaranteed, there may be a chance that the data that gets returned to the client is incorrect or incomplete. Thus, it is better to upload the data, then pull the data, versus uploading and returning the data with the same function. This design choice aligns with the Single-Responsibility Principle (SRP) which states "A module should be responsible to one, and only one, actor."

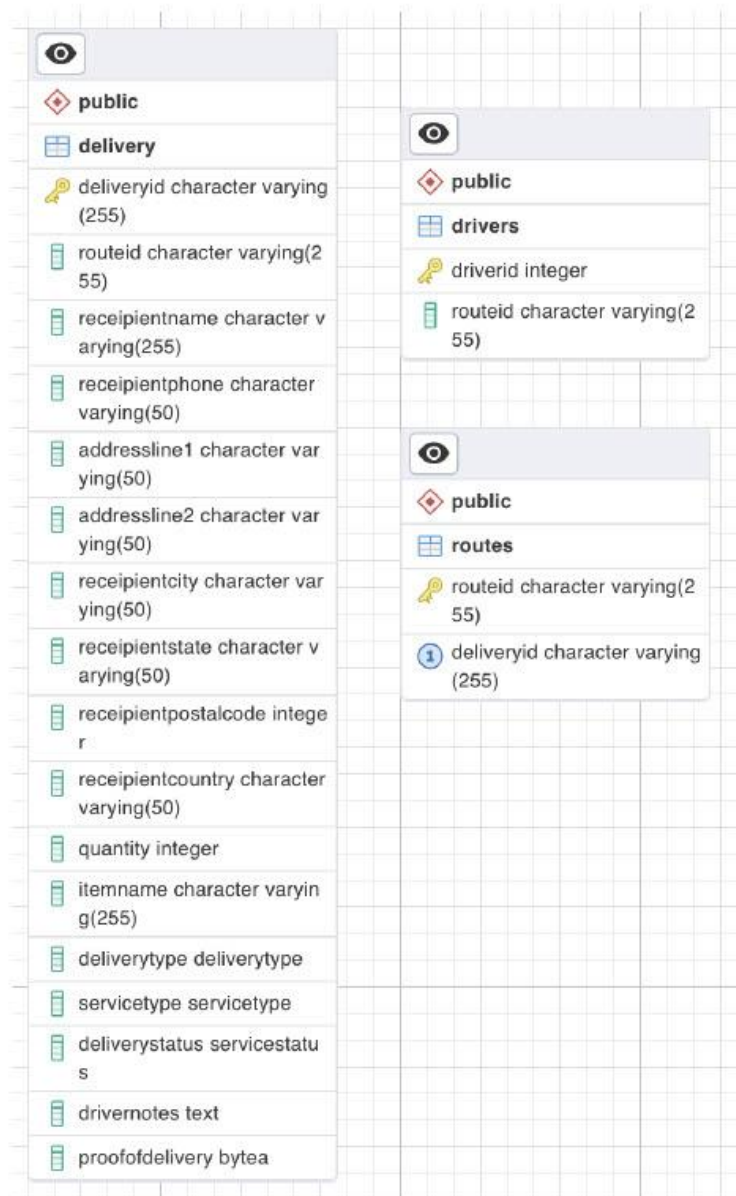
Database Connections and Transactions

As previously mentioned, database connections are handled through a utility class that follows the Singleton pattern. Doing so ensures that there are no zombie connections which will use necessary resources on the backend, and on the DBMS. Following the singleton pattern will also allow for connections to be auto-closing if there is no additional transactions to be executed. One limitation of this approach is that if multiple connections need to happen synchronously (which may happen if there are over 100 clients requesting or pushing data at a time), transactions between the back-end and front-end may see a small delay. This can be avoided through the use of a connection

pool through `psycopg2`, but will require additional checks to ensure the pools are closed at the end of their tasks.

Database transactions are largely handled by the data objects using pre-formatted SQL strings. The purpose of the `DBTransactions` class is to actually execute and commit transactions from each data object. There is no formal data validation in the transactions class.

Database



Front-End Setup Guide

Welcome to the Furniture Delivery App front-end development setup guide. This document provides step-by-step instructions for setting up the development environment and running the application.

Prerequisites

Before proceeding, ensure you have the following prerequisites installed:

1. **Flutter SDK:** The core framework for building the app.
2. **Android Studio:** The recommended IDE for Flutter development.
3. **Dart Plugin:** Required for Flutter development in Android Studio.
4. **Flutter Plugin:** Required for Flutter development in Android Studio.
5. **Git:** Version control system to clone and manage the project repository.
6. **Google Maps API Key:** Required for map functionalities in the app.

Step-by-Step Setup

1. Installing Flutter and Android Studio

1. **Download Flutter SDK:**
 - Visit the [Flutter installation page](#).
 - Download the latest stable release of the Flutter SDK for your operating system.
2. **Extract the Flutter SDK:**
 - Extract the downloaded zip file to a preferred location on your system (e.g., `C:\src\flutter` for Windows).
3. **Add Flutter to the PATH:**
 - Update your environment variables to include the path to the Flutter bin directory.
4. **Install Android Studio:**

- Download and install Android Studio from the [official site](#).
- During installation, ensure that the Flutter and Dart plugins are installed.

2. Configuring Flutter and Android Studio

1. Run Flutter Doctor:

- Open a terminal or command prompt.
- Run the command: `flutter doctor`.
- This command checks your environment and displays a report on the terminal window.

2. Set Up an Android Emulator:

- Open Android Studio and navigate to AVD Manager.
- Create a new Android Virtual Device (AVD) with your preferred configuration.

3. Setting up the Project

1. Clone the Repository:

- Use Git to clone the project repository to your local machine.
- Command: `git clone [repository_url]`.

2. Open the Project in Android Studio:

- Open Android Studio.
- Select 'Open an Existing Project' and navigate to the cloned project directory.

3. Google Maps API Key Configuration:

- Obtain a Google Maps API key following the [official guide](#).
- Place your API key in the Android manifest file (`android/app/src/main/AndroidManifest.xml`) under the `meta-data` tag named `com.google.android.geo.API_KEY`.

4. Running the App

1. Start the Emulator:

- From Android Studio, start the configured Android Emulator.

2. Run the App:

- Select the target device in Android Studio.
- Click the 'Run' button (green triangle) or use the shortcut `Shift + F10`.

3. Debugging and Hot Reload:

- Make changes to your code.
- Use Hot Reload (lightning bolt icon) to instantly view the changes in the emulator without restarting the app.

Additional Notes

- **Dependencies:** Ensure all dependencies are correctly listed in `pubspec.yaml` and run `flutter pub get` in the terminal to fetch them.
- **Environment Variables:** If the project uses any environment variables (like API keys or secret tokens), ensure they are correctly set up in the development environment.

This guide should provide you with a comprehensive setup for starting front-end development on the Furniture Delivery App. For more specific instructions or troubleshooting, refer to the official Flutter documentation or contact the project maintainers.

Back-End Setup Guide

Welcome to the Furniture Delivery App back-end development setup guide. This document provides step-by-step instructions for setting up the development environment and running the server.

Prerequisites

Before proceeding, ensure you have the following prerequisites installed:

1. **PGAdmin4:** GUI to interface with Postgres Database (<https://www.pgadmin.org/download/>),
2. **Docker:** Application containerization system (<https://www.docker.com/products/docker-desktop/>),
3. **Docker-Compose:** Toolchain that helps interface with Docker and simplifies running containers and their setup (<https://github.com/docker/compose>),
4. **Python 3.11:** The language in which the back-end is implemented (<https://www.python.org/>),
5. **Pyenv:** Python environment utility, separate development workspace into Python Environments (<https://github.com/pyenv/pyenv>).
6. **Homebrew:** Only required for Mac users (<https://brew.sh/>),
7. **Postman (Optional):** Use to test API endpoints (<https://www.postman.com/>).



Pyenv is not natively supported by Windows, there is a separate Github repository that works around this. <https://github.com/pyenv-win/pyenv-win>

Step-by-Step Setup

1. Environment Setup

If you are developing on a mac, open your terminal and run the following command:

```
<project root>/scripts/core.sh -c
```

The [core.sh](#) script contains many useful scripts that help automate things such as first-time setup, running the server, installing dependencies, amongst some other features.

After running the above command, you will now be able to run the script by doing:

```
ukay -h
```

The ukay command is added as an alias to your `~/.zshrc` file.



Unfortunately, this script does not run on Windows or Linux machines. For those, you will have to perform a manual setup for Python and Pyenv.

Once all software is installed, cd to the project root and run the following:

```
pyenv install 3.11.0;  
pyenv global 3.11.0;  
pyenv virtualenv ukay-delivery-backend;  
pyenv activate ukay-delivery-backend;  
pip install -r requirements.txt;
```

This code block will create a Python environment with Python 3.11, activate the environment, and install all the required modules for the server.

2. Running the App

- **Start the Database:**
 - Navigate to the `<project root>/docker` directory
 - Execute the following command:

```
docker-compose up -d
```

This command will pull a Postgres Docker image, and run the container. The default credentials are found in the compose.yaml file.



Once the container is running, enter into the database using PGAdmin and run **all** of the queries that are in the <project root>/scripts/init.sql file.

- **Start the Server:**
 - If you are on a Mac, run the following:

```
ukay -r
```

For all other platforms, run:

```
cd <project root>/app  
python .
```

The server will run using the following address:

<http://localhost:21000/>

You may view the API Documentation by visiting the following link in your web browser:

<http://localhost:21000/docs>

Additionally, there is a JSON representation of the documentation in <project root>/api/openapi.json

Additional Notes

- **Dependencies:** If any dependencies must be updated, ensure you generated a new requirements.txt file by doing: `pip freeze > requirements.txt`
- **Environment Variables:** If the project uses any environment variables (like API keys or secret tokens), ensure they are correctly set up in the development environment.

This guide should provide you with a comprehensive setup for starting backend-end development on the Furniture Delivery App. For more specific instructions or

troubleshooting, refer to the official FastAPI, Python, and Docker documentation or contact the project maintainers.

Wish List

The following are features that have yet to be implemented, or are partially implemented which are part of Ukey Delivery's original requirements:

- Automatically inform clients that their driver is 30 minutes away,
- Implement a chatting feature so that drivers can communicate with their client,
- Allow drivers to update the delivery status of packages,
- Allow drivers to upload a "proof-of-delivery" photo in order to complete a delivery,
- Deploy the back-end server to a cloud service (**must be an IaaS provider, Firebase will require a significant rework of the backend**),
- Secure the back-end server,
- Implement data caching on the front-end, and data syncing for cached data,
- Implement users with different roles (such as administrator, driver, dispatch, etc).

REFERENCES

- DigitalOcean. (n.d.). S.O.L.I.D: The First Five Principles of Object-Oriented Design. Retrieved from
<https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design>
- Mozilla Developer Network. (n.d.). Model-View-Controller (MVC). Retrieved from
<https://developer.mozilla.org/en-US/docs/Glossary/MVC>
- Google Cloud. (n.d.). What is Microservices Architecture? Retrieved from
<https://cloud.google.com/learn/what-is-microservices-architecture>
- IBM. (n.d.). Client/Server model. Retrieved from
<https://www.ibm.com/docs/en/cics-ts/6.1?topic=programs-clientserver-model>
- Oracle. (n.d.). Data Access Object. Retrieved from
<https://www.oracle.com/java/technologies/data-access-object.html>

Apple Developer Documentation. (n.d.). Managing a Shared Resource Using a Singleton.

Retrieved from

<https://developer.apple.com/documentation/swift/managing-a-shared-resource-using-a-singleton>

GeeksforGeeks. (n.d.). Builder Design Pattern. Retrieved from

<https://www.geeksforgeeks.org/builder-design-pattern/>