

# **CS307 – Software Engineering Design Document**

## **Delivery and Order Management System (DOMS)**

**Team 15**

Dinesh Gajwani, Kunal Goyal, Hemil Desai, and Shivank  
Tibrewal

# Table of Contents

- i. Purpose
- ii. General Priorities
  - 2.1. Performance
  - 2.2. Usability
  - 2.3. Security
  - 2.4. Hosting, Development and Scalability
  - 2.5 Appearance
- 3. Design Outline
  - 3.1. Overview and Components
  - 3.2. Architecture
  - 3.3. Backend API
  - 3.4. Database Server
  - 3.5. Web Application
  - 3.6. Mobile application for Drivers
- 4. Design Issues
- 5. Design Details
  - 5.1. Components
    - 5.1.1. Web app for Stores and Administrator
    - 5.1.2. Backend API
    - 5.1.3. Database Server
    - 5.1.4. Mobile Application for Drivers
  - 5.2 Class Design
  - 5.3. Sequence Diagrams
  - 5.4. UI Mockups

## **PURPOSE**

In our current society, delivering customers right to their doorstep significantly raises the value of any store. Although local stores would like to provide delivery, they often do not have the means or capital to do so. Our solution will help local store owners to deliver their orders through an automated delivery and ordering system. Not only will this give the local stores an online identity, but it will also help them compete against big box online retailers.

The following items the major areas of functionality within the application. These areas capture the key user facing elements of the application.

- **Account Management:** Store owners, drivers and administrators must be able to manage their own and other user accounts for which they have the appropriate permissions. 1)Store owners manage their own accounts and dispatch delivery requests. 2)Administrators have the ability to perform the tasks needed to maintain the service, including managing deliveries and user accounts(Drivers, Store owners) . 3) Drivers manage their own accounts and may accept or reject delivery requests.

- i. **As a administrator, I should be able to:**

- i. To add store owners
  - i. Add store owners.
  - ii. Delete store owners.
  - iii. Modify store owners.
- ii. To manage users
  - i. Add users.
  - ii. Delete users
  - iii. Modify user data.
- iii. To manage drivers
  - i. Add drivers.
  - ii. Delete drivers
  - iii. Modify user drivers.
- iv. Locate active drivers on map
- v. Generate a list of all active deliveries.
- vi. View feedback by users

**ii. As a Store owner, I should be able to:**

- i. Add orders to the system
- ii. Reset my password if I need to.
- iii. To manage drivers
  - i. Add drivers.
  - ii. Delete drivers
  - iii. Modify driver details.
- iv. Assign orders to available drivers - automatically and manually.
- v. Locate working drivers
- vi. Easily use the system with existing ordering mechanisms.
- vii. View feedback by users

**iii. As a Driver, I should be able to:**

- i. Accept and reject delivery
- ii. Reset my password if I need to.
- iii. See the delivery address on the map
- iv. Get an ETA
- v. See the best route to get to the address
- vi. Complete a delivery

- **Delivery Management:** The algorithm for scheduling and managing delivery will be a first come first serve algorithm. This will be implemented using a Queue - FIFO data structure. If time permits, another algorithm will be implemented on top of the FIFO algorithm. This algorithm will group deliveries based on preferences and proximity. All the deliveries in a neighbourhood will be grouped together and dispatched to a single driver. Both the algorithms will make delivery scheduling efficient and feasible.
- **Customer Feedback Management:** Since the customers are the ones who will receive the deliveries, there needs to be a mechanism for them to provide feedback based on the service. This will be implemented by generating a unique link for each order. This unique link will then be sent to the customer to provide feedback, and that feedback will be matched back to the driver who made the delivery.
- **Data Analytics:** For each store, the store owner will have access to a store dashboard that will display all the data as required by the store owner. This

will display statistics like orders per month, revenues per month, drivers used by the store, driver ratings, etc. This will help the store keep a track record of their transactions on a monthly basis.

## **GENERAL PRIORITIES**

The general priorities of the application are focused on ensuring the success of the application by prioritizing the user experience, and providing quality functionality to the store owners and drivers who will use the software.

### **Performance**

To ensure a seamless and fluid experience for store owners, the server response time should be less than 500ms and ideally less than 250ms. On the client Javascript execution should be less than 1s (with a variable timeout). The overall goal for performance is to provide a responsive environment for both the store owner and driver.

### **Usability**

The application must provide a clean and clear interface for the necessary operations a store owner or a driver must perform. The presentation of options must be kept simple to avoid confusion which would only slow down the service and detract customer experience.

### **Security**

Security is an important feature for this application. The consequences of security breaches are relatively low, since no sensitive information such as payment information is stored. Account passwords will be encrypted using standard password encryption techniques; if time permits, OAuth will be utilized for authentication.

Interaction between clients and server shall be encrypted using SSL/TLS to avoid basic session hijacking attacks. Taken as a whole, we adopt the philosophy of utilizing simple, easy to implement security measures that provide a high benefit to cost ratio. A provably secure system is not required and is not worth the effort needed.

### **Hosting, Development and Scalability**

The application will be developed and deployed on Amazon AWS, which provides tools such as automatic load balancing which will help us in scaling the application in the future. AWS has a proven track record and is one of the most reliable services available for a nominal cost.

**Appearance**

The application should appeal to the target audience in a positive and engaging way. The store owners should have a convenient design which should be organized to help in peak hours. The mobile apps for the driver should have a clean UI with fresh colors which would promote a encouraging work environment. Information to both, should be limited to bare minimum, so that the application is easy to use.

# **DESIGN OUTLINE**

## **Overview and Components**

The system will implement a client-server architecture with the following components

- backend API
- Database Server
- Web application
- Mobile application for Drivers

## **Architecture**

The application will use a common client-server paradigm to provide separation of concerns between the client and server. This will facilitate easier division of effort between the development team members and allow the UI and backend components to evolve in a flexible manner.

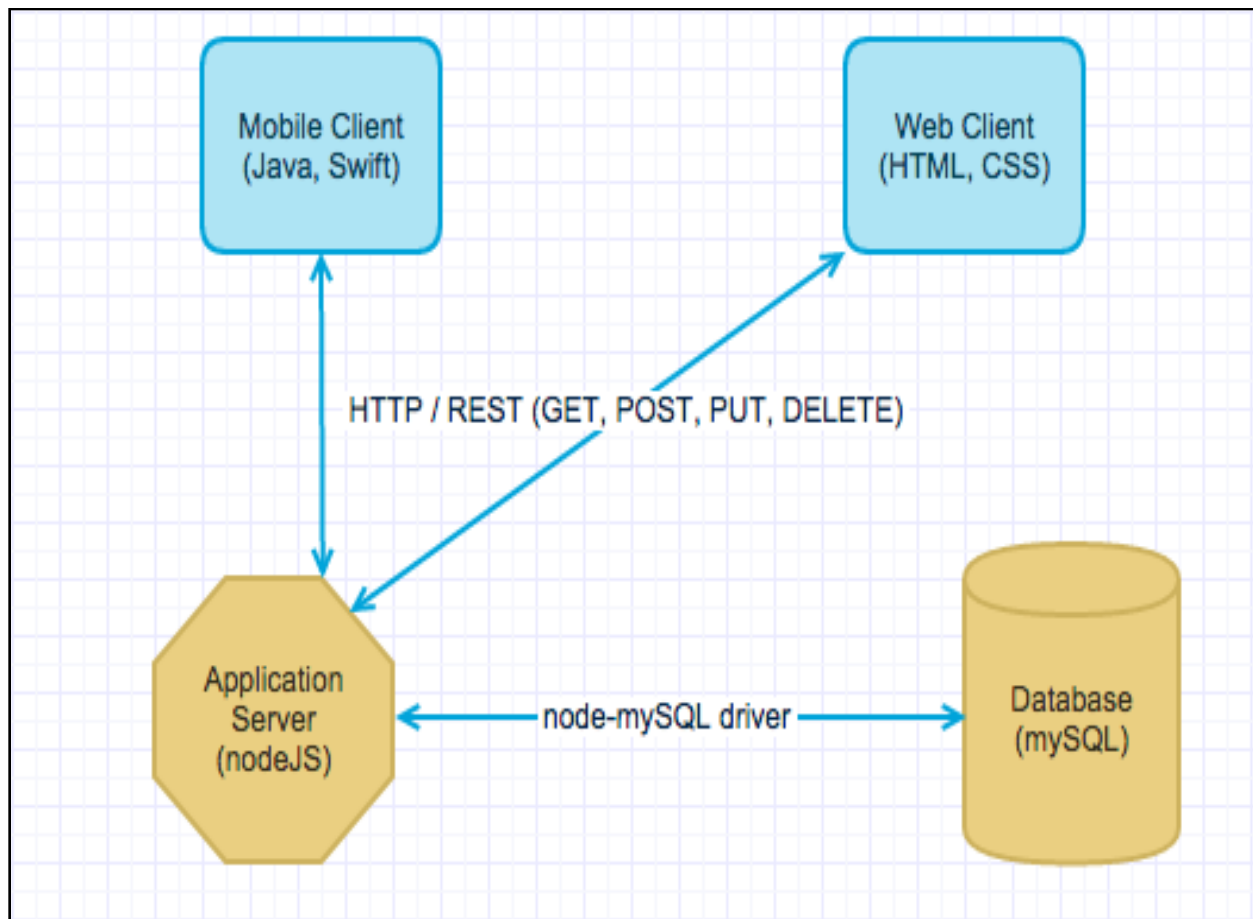


Figure 1: Platform Architecture Diagram



## **Backend API**

This API will be implemented using Node JS and express. The backend API will be responsible for connecting requests from store owners and drivers. It will be able to send HTTP requests or AJAX requests from store owners and accept the HTTP requests or AJAX requests at the driver end.

## **Database Server**

All of the user and class management functionality will be located within the database server component. The database server will be responsible for connecting the application to the database. It will contain all of our store and driver information linking everything together.

## **Webapp for Stores**

The web application will use languages like HTML, javascript and CSS and frameworks like AngularJS for the front-end. The web application will have an admin dashboard as one of the components. This will be used by the administrators to monitor available drivers and pending requests along with complete overview of all stores and drivers. The web application will also have a component for store owners. This will act as the host delivery platform for the store owners to send the order details to the drivers and monitor the order progress and feedback. A component for drivers to register for the service will also be available. A primary goal for the web application is to be simple, easy to use, visually attractive, and responsive to stores' needs. The web application will interact with the backend API for various order related tasks and for tracking their drivers and other information regarding them.

## **Mobile application for Drivers**

The mobile application for drivers will be built as an Android application. This will act as the platform for drivers to receive order details and customer information regarding the delivery. The mobile application should have good UX that makes it easy for the drivers to handle all their tasks. The application will interact with the backend API for the receiving and accepting orders and for updating their status at every step of the delivery process.

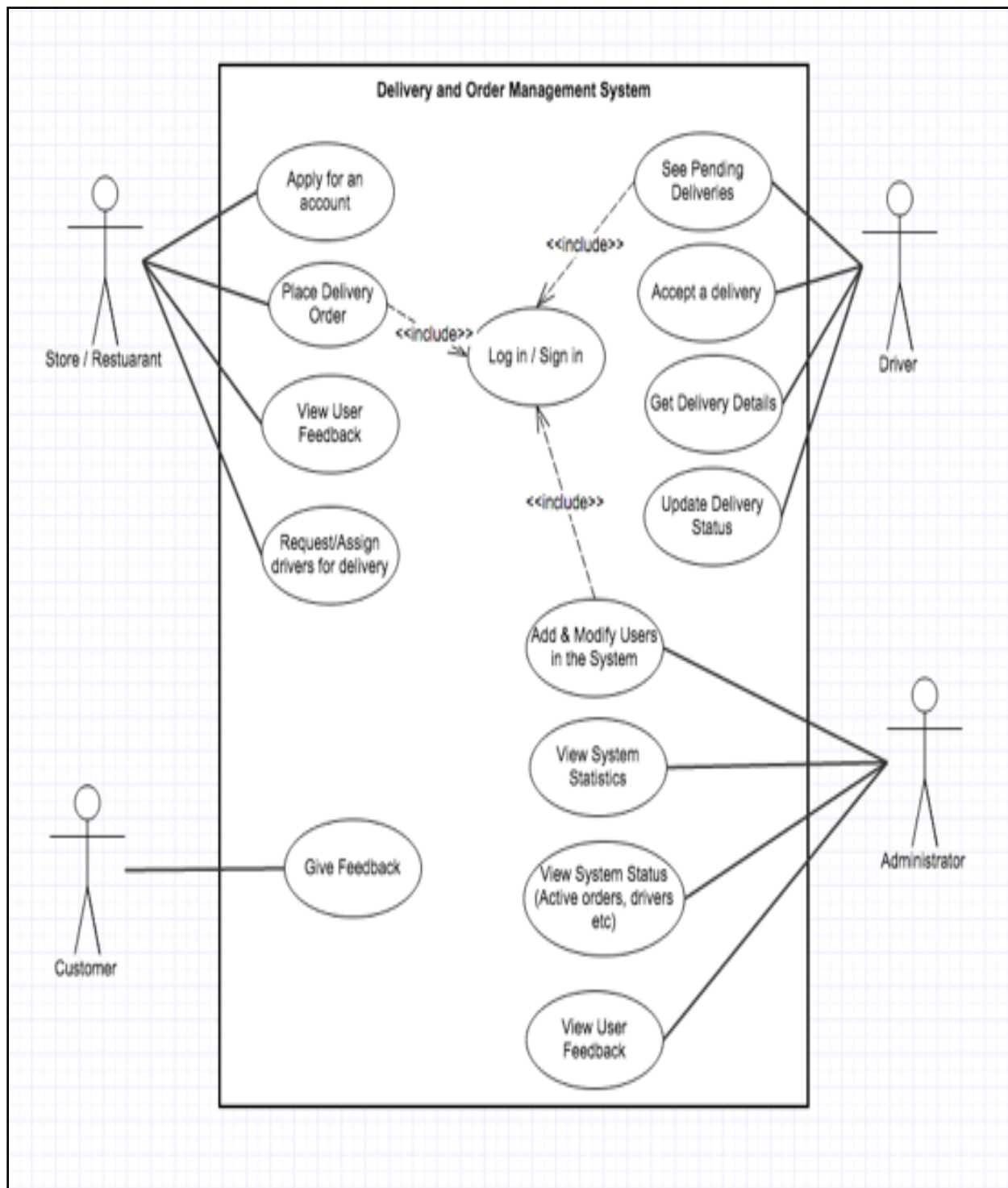


Figure 2 : Use Case Diagram

# **DESIGN ISSUES**

The following design choices were made as part of the planning process:

- **Design Issue 1 - Designing the backend for the platform:**

- Option 1 - nodejs and express
- Option 2 - Ruby on Rails
- Option 3 - Django
- Option 4 - Laravel

We decided to choose option 1. Firstly, nodejs shines in real time updates. So Node JS allows us to send real time updates to drivers, store owners and customers with ease. Secondly, Node JS uses javascript that allows us to unify the language and data format (JSON), which makes it easier to reuse developer resources. Thirdly, Node JS has a very large and growing community which makes it easier to get support. Using express is also beneficial as it is a lightweight framework and allows a lot of flexibility.

- **Design Issue 2 - Designing the front end web app:**

- Option 1 - angularjs
- Option 2 - React
- Option 3 - backbone
- Option 4 - ember
- Option 5 - Templating engine

We decided to choose option 1. Angularjs is extremely modular and lightweight. Angularjs also seamlessly integrates with Node JS. Using angularjs allows us to keep the backend and frontend separate, and this allows to reuse our API across both web and mobile applications. Angularjs also has a very large and growing community, and that makes it easier to get support.

- **Design Issue 3 - Database selection:**

- Option 1 - mySQL
- Option 2 - Postgres
- Option 3 - SQLite
- Option 4 - mongodb
- Option 5 - rethinkdb

We decided to choose option 1. We will use the node-mysql driver for connecting the database to the backend API. MySQL is extremely popular and all our team members are familiar with it. It is agile and scalable, and has very good development and operational tools. It is also easy to make relational queries using MySQL.

■ **Design Issue 4 - Authentication and Authorization:**

- Option 1 - email/ password local store
- Option 2 - Login with Facebook
- Option 3 - Login with Twitter
- Option 4 - Login with Google

We decided to choose options 1 and 2 and integrate them together to get a common user object. Facebook is very common and most local stores have facebook pages promoting their stores. Also, most consumers and drivers use facebook. Using their facebook data allows us to add feedback functionality into our platform with ease. Local data store is for those who do not use facebook. The local information can be integrated with Facebook details and create a unified user object.

■ **Design Issue 5 - Mobile application:**

- Option 1: iOS
- Option 2: Android
- Option 3: Ionic
- Option 4: Apache Cordova

We decide to choose option 2. Most of our team members are familiar with android and that makes it easy to develop the mobile application. Android is Java based, and all of our team members are familiar with java. It is also easy to get support while developing android apps.

■ **Design Issue 6 - Payment Platform:**

- Option 1 - Stripe
- Option 2 - WePay
- Option 3 - Braintree
- Option 4 - PayPal payment buttons

We decided to choose option 2. Braintree's v.zero allows us to integrate Paypal, Android Pay, Apple Pay, Venmo, coinbase, cards easily. This allows for greater flexibility for accepting and making payments. Braintree is a paypal

owned company, which makes it easier for us to get support due to paypal's large community. The payment gateway should handle two purposes. We should be able to accept commission based on every order from the store on a monthly basis. We should also allow the store owners to pay the drivers for their services. For development purposes, we will use the braintree sandbox. During deployment, we will use a production account to handle real payments

■ **Design Issue 7 - Delivery Scheduling Algorithm:**

- Option 1 - First come First serve
- Option 2 - Grouping algorithm

We decided to choose option 1 due to the time constraints we face.

Implementing a first come first serve algorithm for deliveries is efficient and appropriate in the present scope. This will be implemented using a FIFO data structure. If time permits, we will use option 2 on top of option 1 that will allow us to group deliveries based on proximity (all orders in a neighbourhood will be grouped and sent to a single driver).

# **DESIGN DETAILS**

## **Components**

### **i. Webapp for Stores and Administrator**

A browser based front end to the system, to provide ease of use in management of the system. The client will be implemented using HTML, CSS, and JavaScript. As supplements, we will use Angular libraries to simplify programming. Frameworks like bootstrap will be used to develop a better UI. Also, libraries like *Materialize* and *Bootstrap Dashboard* themes seem to be working well with our concept. The majority of the content will be delivered to the client in response to standard HTTP requests to the backend API.

### **ii. Backend API**

The backend API will be developed to interact with a server (preferably cloud) to serve content to the client. The initial development server will be running on a local machine. The backend will be implemented using Node.js. Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. It will use the Express framework to handle general web server functionality and data management. Express Router will be used to create routes. Express will also be used to create our API. We will use passport to implement authentication and authorization for both our web apps and API (including login with facebook). We will be using different middlewares to process the HTTP requests and send responses. Morgan will be used to log errors, Flash will be used to display alerts, body-parser will be used to parse request body and cookie parser will be used to parse cookies. We will add more middlewares based on requirements. Additionally, we will use multiple JS libraries to aid in development, testing, and production efficiency.

### **iii. Database Server**

A single highly performant physical or virtual server to store static data such as stores, drivers and orders. We will use MySQL because it is easy to use, secure, scalable, fast and inexpensive.. Adapters and ORM libraries are available for Node JS for MySQL, so no significant compatibility issues are expected and integration with the backend API will be easy.

#### **iv. Mobile App for Drivers**

The mobile app will be built in Android. The mobile app will simply make API calls to implement different functionality. We are also considering using Xamarin or Ionic framework, which will also us to build Android and iOS apps at the same time, without writing separate code for both.

## Class Design

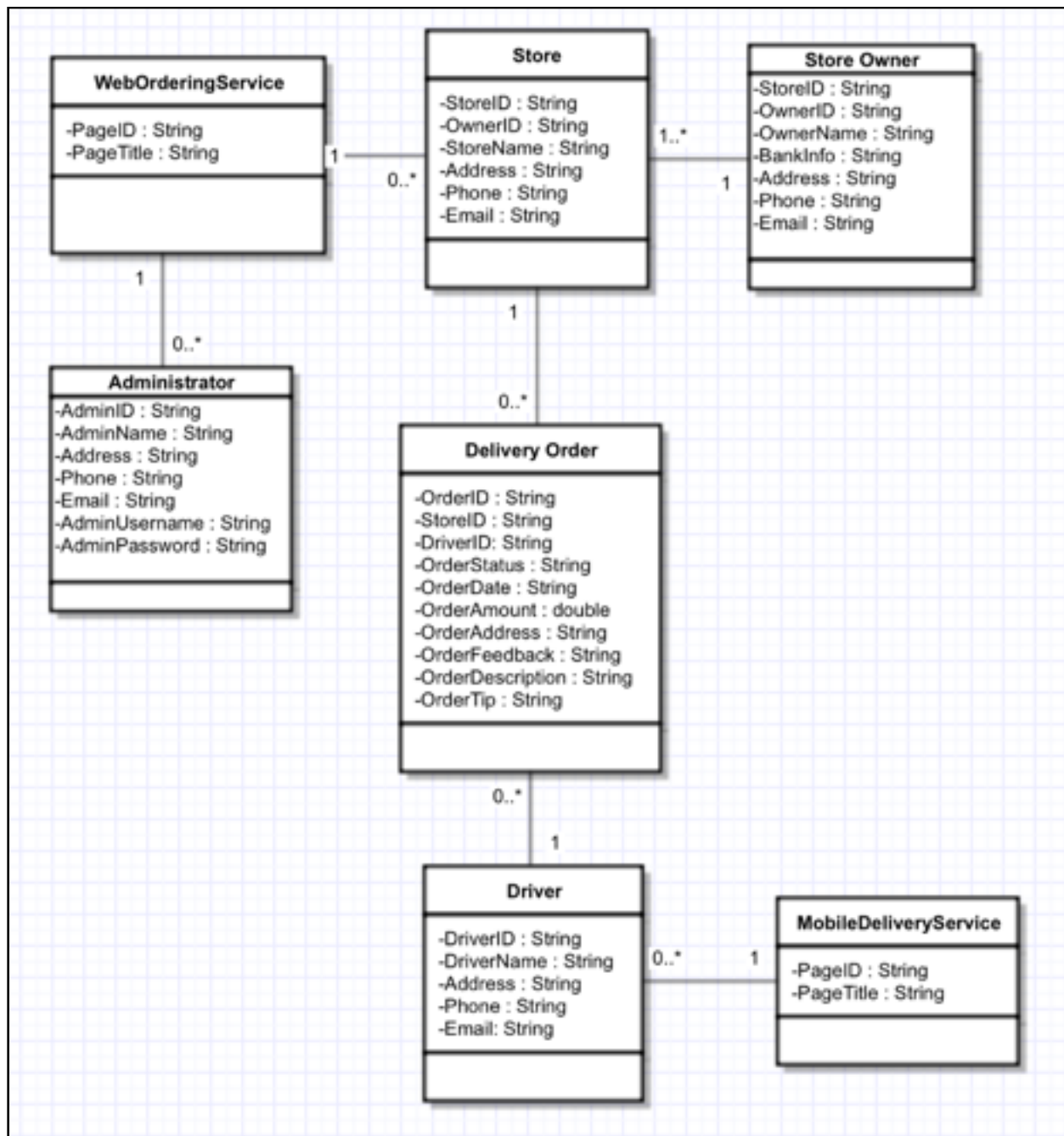


Figure 3. Class diagram depicted in terms of entities-relationships and all related attributes if it were to be implemented as database tables.



## **Description of classes and models:**

### **Store**

A store represents everything related to a store in the database with each store having a store owner associated with it with an unique OwnerID. A store can have multiple delivery orders associated with it and can place these orders on the Web Ordering Service.

### **Store Owner**

A store owner represents everything specific to a store owner to make payment transactions. It is uniquely identified for each Store using an OwnerID along with all the personal contact information of the Store Owner like BankInfo, address, phone number, etc.

### **Delivery Order**

A delivery order represents everything related to an order created from a store on the application. The orders are uniquely identified with their OrderID and are made with Stores along with all the delivery information. Drivers can accept orders and update the status of the delivery on the order.

### **Driver**

A driver represents everything to a driver who is present inside the list of drivers in the database. A driver is uniquely identified with its DriverID and this class contains all other contact details of the driver including address, phone number, email, etc. A driver can access order information and accept/reject orders. After accepting the offers, the driver is also responsible for updating the status of the order continuously.

### **Mobile Delivery Service**

A mobile delivery Service holds all the content that a driver needs to access/update delivery orders sent to it. It is an interactive UI for the drivers to accept/reject orders and view their details in order to successfully complete the delivery.

**Web Ordering Service**

A web ordering Service holds all the content that a store needs in order to manage orders and create orders for delivery and monitor them at the same time.. It is an interactive UI for the stores to create orders and view their details in order to successfully ensure that the delivery is made with customer satisfaction.

Administrator can also view all the data in the database using the Web ordering service.

**Administrator**

An administrator is uniquely identified with an AdminID along with all other details required to identify an Admin like Address, phone, username, etc. Admin has the access to Web Ordering Service where he/she can monitor all the orders along with all the data in the database including stores, orders and drivers.

## Sequence Diagrams:

### **Store Employee Sequence Diagram:**

A store employee places a delivery order in the system and interacts with the system until the delivery is completed.

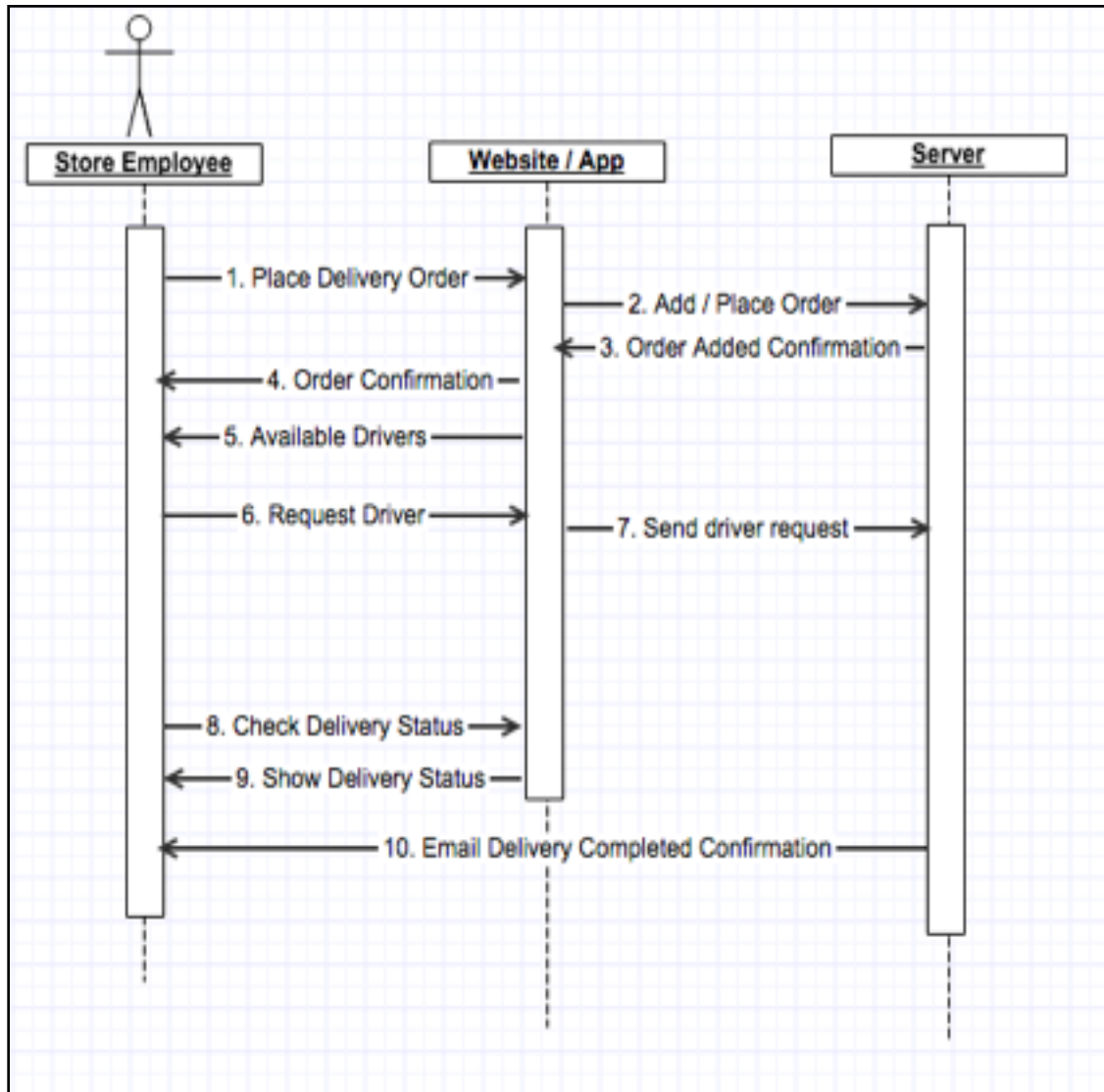


Figure 4. Store Employee Sequence Diagram

### Driver Sequence Diagram:

A driver can check available delivery orders, accept delivery orders and update them.

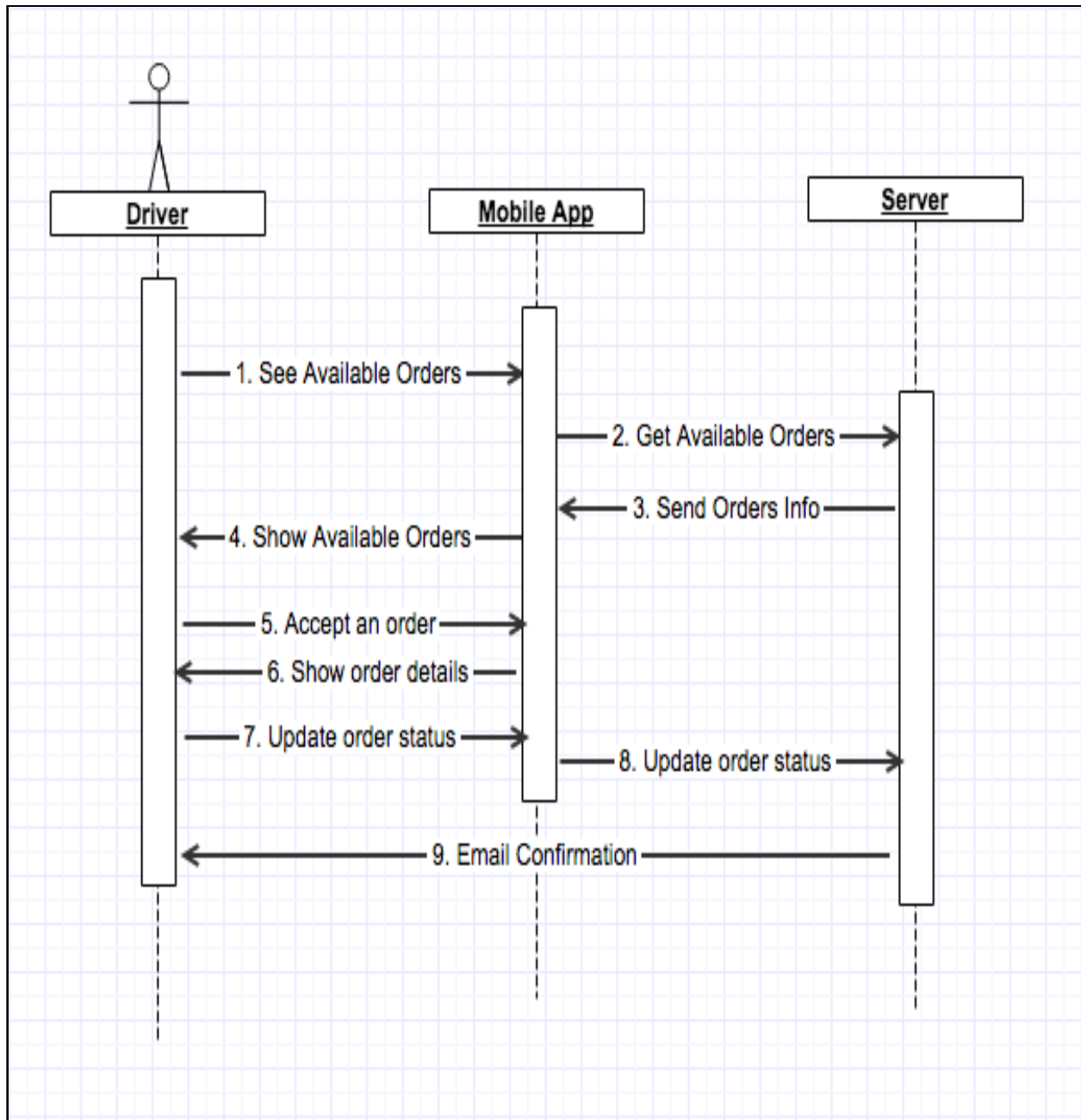


Figure 5. Driver Sequence Diagram

### Customer Sequence Diagram:

A customer who receives a delivery is provided a link that requests a feedback from them.

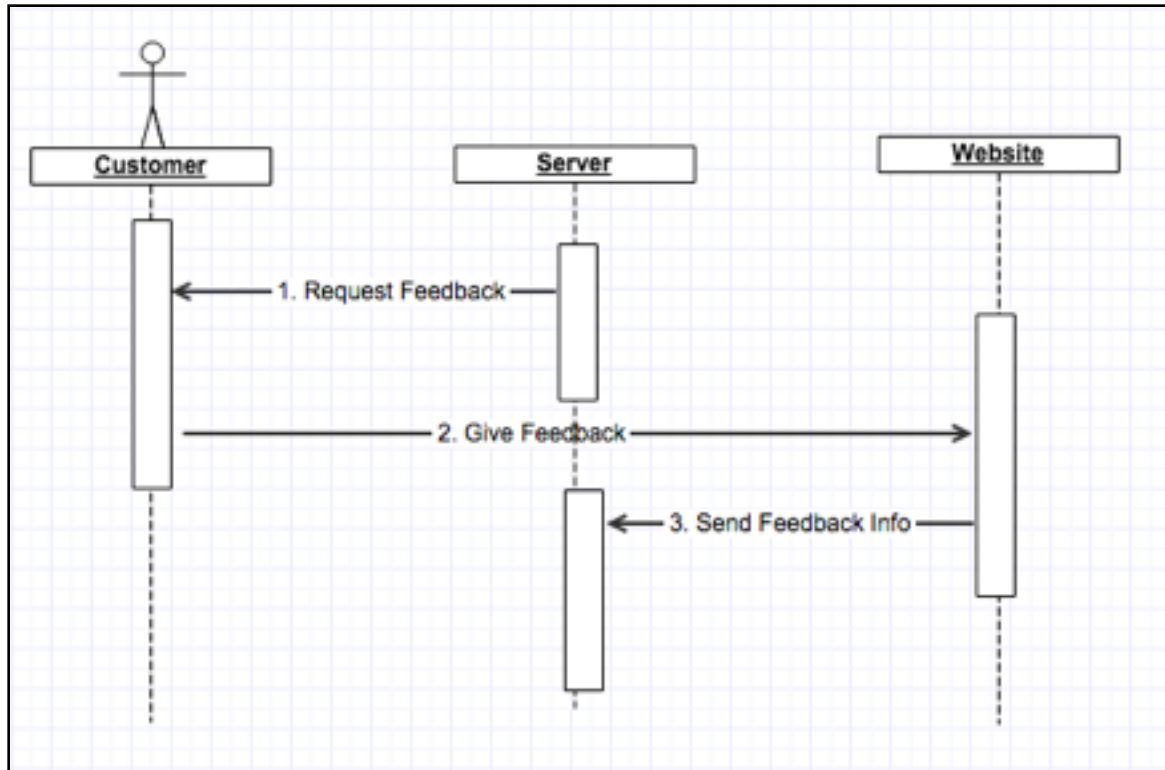
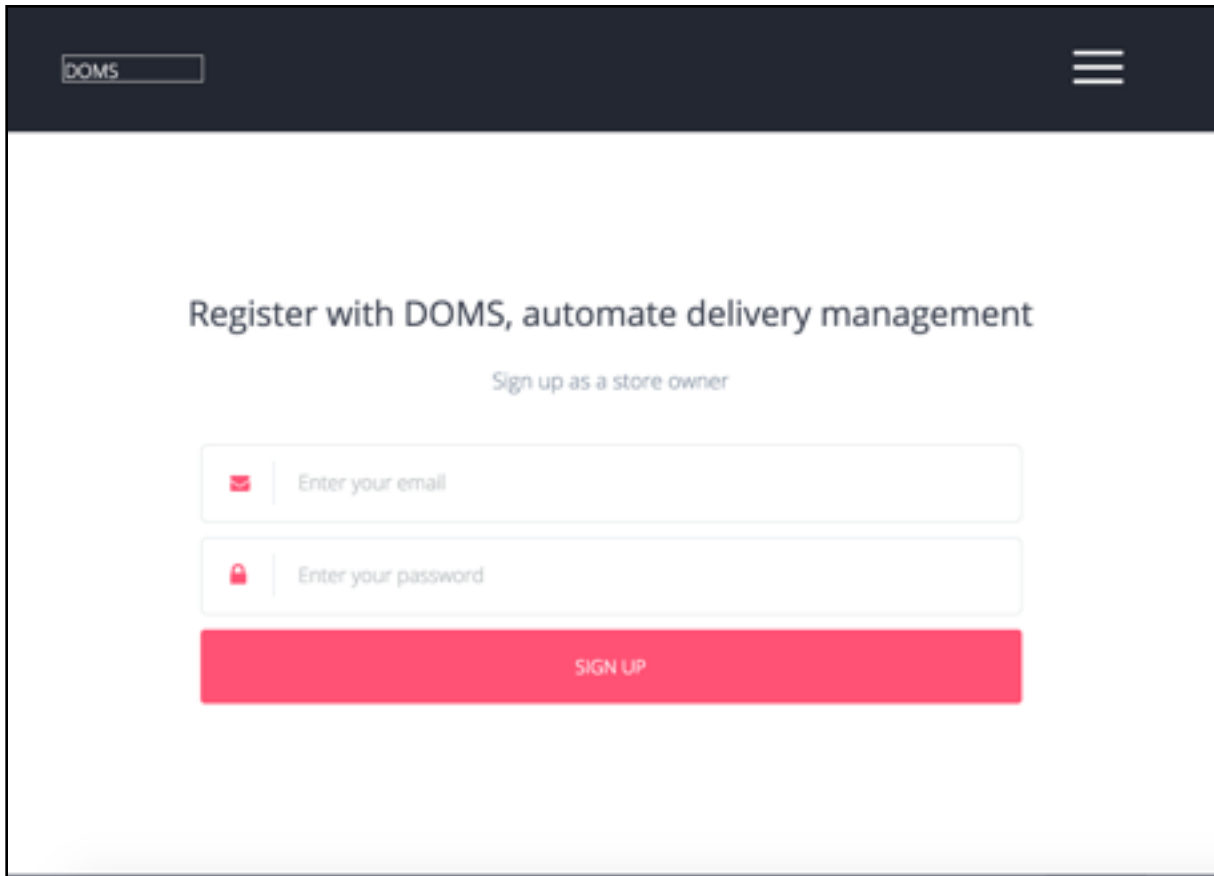


Figure 6. Customer Sequence Diagram

# Graphic User Interface Mockups

## 1. Registering a Store:



The mockup shows a registration form for DOMS. At the top, there is a dark blue header with the 'DOMS' logo on the left and a hamburger menu icon on the right. The main content area is white and contains the following elements:

- Headline: "Register with DOMS, automate delivery management"
- Sub-headline: "Sign up as a store owner"
- Email input field: A white box with a red envelope icon on the left and the placeholder text "Enter your email".
- Password input field: A white box with a red padlock icon on the left and the placeholder text "Enter your password".
- Sign Up button: A solid red rectangular button with the text "SIGN UP" in white, centered.

Figure 7: Signup as a store owner

## 2. Mobile App for Driver - Updating Status.

A hand-drawn sketch of a mobile app interface for updating order status. The interface is contained within a rectangular frame. At the top, the title "Update Order Status" is written in a cursive font, with a hamburger menu icon (three horizontal lines) to its left. Below the title, there are three rows, each separated by a horizontal line. The first row contains the text "Order Ready" followed by an empty radio button. The second row contains the text "Picked up" followed by a radio button with a dot in the center. The third row contains the text "Delivered" followed by an empty radio button. At the bottom of the frame, there is a rounded rectangular button labeled "SAVE".

Update Order Status	
Order Ready	<input type="radio"/>
Picked up	<input checked="" type="radio"/>
Delivered	<input type="radio"/>

SAVE

Figure 8: Driver Updating Status from App.

### 3. Active Orders visible to Drivers in Mobile App

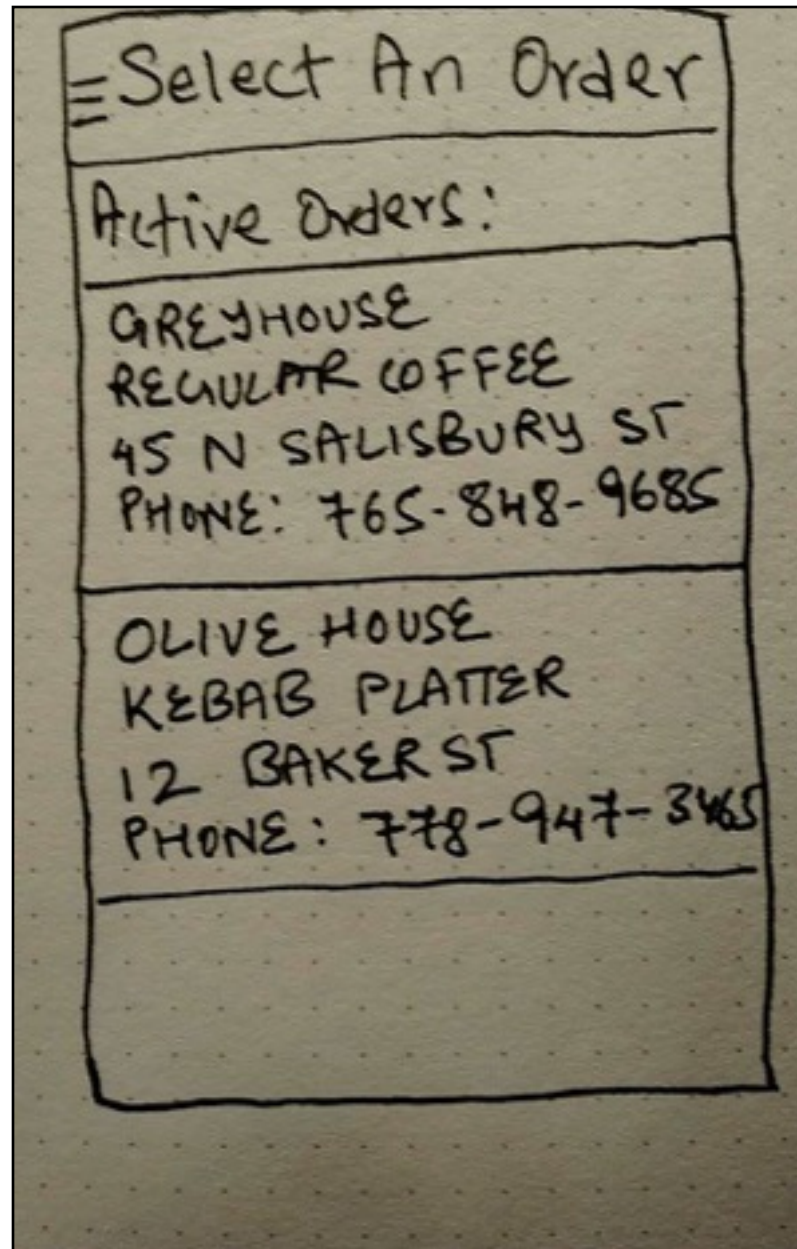


Figure 9: Active Orders visible to Drivers



#### 4. Placing an order for delivery

--DOMS--

### PLACE ORDER FOR DELIVERY

ITEM NAME:

ITEM TYPE:

DELIVERY DETAILS:

FIRST NAME:

LAST NAME:

PHONE :

ADDRESS :

EMAIL :

AMOUNT :

Figure 10. Placing an order for delivery

## 5. Checking active orders

-- Doms --

-- GREENHOUSE --

ACTIVE ORDERS:

ORDER ID: 1234 SHIP TO: 3463 BAKER ST, APT 11, WEST LAFAYETTE, IN 47906 KUNAL HOLMES TOTAL: \$5.00 ITEM: 1 LARGE COFFEE	ORDER STATUS: PICKED UP DRIVER INFO: WATSON PHONE: 765 974 0221
ORDER ID: 1235 SHIP TO: 45 N SAUSBURY ST, APT 10, 47906 TOTAL: \$10.99 ITEM: 2 CAPS	ORDER STATUS: ORDER READY

Figure 11. Checking active orders.