Software Design

Module: CS5721

Lecturer: JJ Collins

Bin Management System

Submitted by

Dharaneesh Venkatesh	19018304
Jansirani Subburam	19166125
Kailash Muralidharan	19116608
Kaustab Basu	19005946

Table of Contents

1. Project Description

1.1 Overview

Smart Bin is a dedicated Bin Management solution for cities and businesses to digitize waste bin inventories, manage bin distribution and make data-driven decisions. It is totally sensorless, thus doing away with the need to monitor real-time data on fill-level. Smart Bin allows the administrator at the Bin Company to build a detailed bin inventory, optimize bin locations, schedule and plan trips with shortest possible paths to reach optimum truck capacity. It allows customers registered with the Bin Company to raise requests to pick their bins with the click of a button. It allows Truck Drivers to view their daily assignments for bin pickups, start and complete their trips.

1.2 Business Description

The Bin Management Solution proposed by Smart Bin involves creating a web-based application for use by registered customers, truck drivers and the administrator of Bin Company. To start using the application, the customers and truck drivers will be required to register for a one-time account on the website. The application is expected to function with high and lightning-fast exchange of data records, thus mandating the need for a remote database management system to handle data management.

To meet real-time, continuous demand from customers, the application must have high availability and scalability. In order to achieve this, the web-based application is required to be hosted in a dedicated host on the Cloud and also be capable of load-balancing.

The application will have an online payment feature allowing customers to view and pay their monthly bills using their Debit or Credit cards. A trusted third-party merchant integration is required on the application to make sure that the transactions are successful, enhancing trust and confidence with the customers.

The application will sport a unique analytics hub, providing administrators of the Bin Company with statistics and reports, allowing them to make data-driven decisions. This will enable them to plan efficient waste collection routes, calculate fuel to collection costs and adjust bin distribution based on patterns observed in customer requests. A tried and tested machine learning based solution, offering predictive analytics is suggested to completely realize the above, allowing the Bin Company to make intelligent decisions.

1.3 Case Tools

1.3.1 Integrated Development Environment (IDE)

Visual Studio Code

Visual Studio Code is a source-code editor developed by Microsoft for Windows, Linux and macOS. It includes support for debugging, embedded GitHub version control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is highly customizable, allowing users to change the theme, keyboard shortcuts, preferences, and install extensions that add additional functionality. The source code is free and open source and released under the permissive MIT License.

1.3.2 Version Control

GitHub

GitHub was used for version control and collaboration during the development process. A new repository for the project, **'smartbin'** was created and all members of the team were added as collaborators. The Visual Studio Code IDE and GitHub Desktop application were used by team members to connect to the repository and contribute from their systems directly. The platform was also used as Scrum Board for tasks allocation and status tracking during sprint plans.

1.3.3 Diagrams Creation

- StarUML
- BOUML
- Lucidchart

2. Software Development Life Cycle (SDLC)

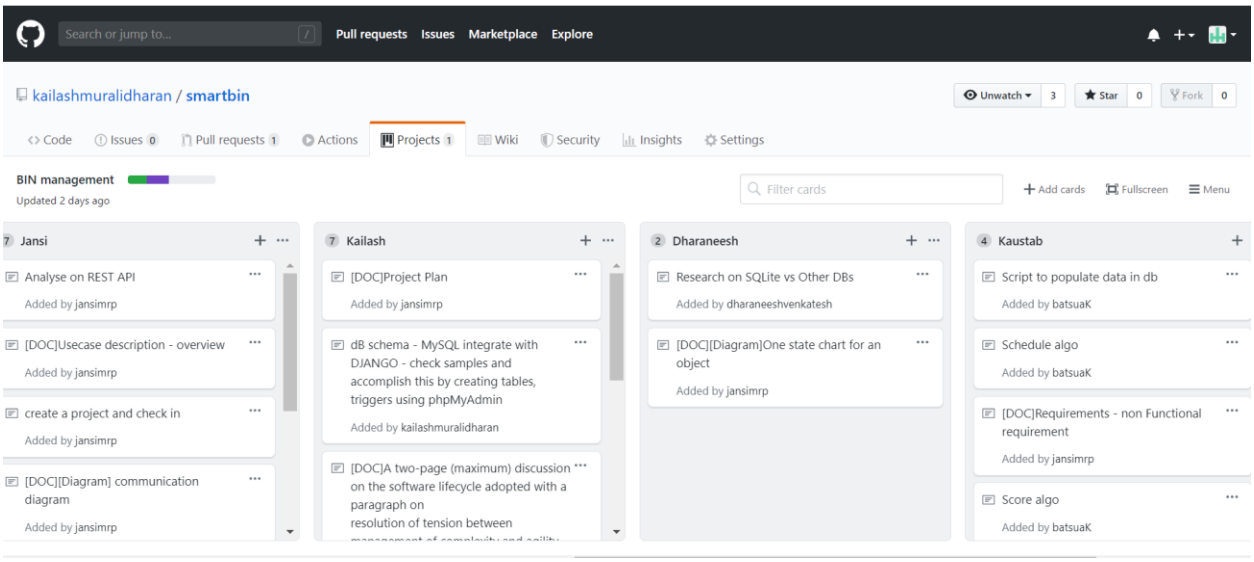
For our project, we adopted the following 5 phases in Software Development Life Cycle: Requirements Analysis, Design, Development, Testing and Support.

Given the nature and constraints of time involved, we considered Agile Methodology to be most suited for the project. Agile focuses more on people doing the work, self-organizing themselves and how they collaborate with each other, adding their respective strengths to the team, ultimately leading to great solutions evolving in an uncertain and turbulent environment. We understood that though the team members had come from varying backgrounds, every one of them had the ability to figure out how they're going to approach tasks on their own and transfer their skills to making progress in the project.

After a dedicated comparison between Scrum and Extreme Programming, we found Scrum to be more suitable to the project. Since we had a good understanding of each team member's strengths, using Scrum

meant that we were able to prioritize tasks and delegate them to the team member who was best suited to solve that. We used GitHub platform for our Scrum Board, for tasks allocation and status tracking. Sprint plans were devised with each sprint carrying out an equal weightage of overall tasks that were originally agreed upon. During the development phase, each week started with a Sprint Planning call, setting objectives and plan for the week ahead, followed by a Retrospective call at the end of the week to discuss progress in results and pitfalls, if any. Apart from this, we had regular, short daily Scrum stand-ups that lasted for 15 minutes, when each team member addressed their plan for that day.

All communications were done through the use of emails, WhatsApp group texts, as well as GitHub Scrum Board. Project code and other related materials were available to all team members through GitHub and Google Drive respectively.



3. Project Plan

3.1 Role Assignment

Table 1

	Role	Description	Designated Team Member
1	Project Manager	Drives sprint planning and review meetings, gets agreement on the project plan, tracks progress	Jansirani Subburam
2	Documentation Manager	Responsible for sourcing relevant supporting documentation from each team member and composing it in the report	ALL

3	Business Analyst/ Requirements Engineer	Gathers functional and non-functional requirements, use case diagrams & description	Kailash Muralidharan
4	Architect	Defines high level system architecture and gets agreement on technology pipeline	Kailash Muralidharan
5	System Analysts	Creates conceptual class model and drives implementation effort	Dharaneesh Venkatesh
6	Designer	Responsible for recovering design time blueprints from implementation	Dharaneesh Venkatesh
7	Scrum Master/ Technical Lead	Leads the implementation effort, handles daily Scrum stand-up calls	Kaustab Basu
8	Programmers	Understand sprint plans, develop features as defined in requirements, perform unit testing and peer code review	ALL
9	Test Manager	Drives coding of automated test cases, tracks defects, prioritizes issues	Jansirani Subburam
10	Dev Ops	Ensures that each team member is competent with development infrastructure, e.g. GitHub, etc.	Kaustab Basu

3.2 Project Schedule

Table 2

Week	Workflow
3	Team set-up, business scenario identification, understanding team strengths, research on existing projects, etc.
4	Requirements gathering, prepare use case diagrams and description
5	Use case realization leading to Analysis Class diagram
6	Decision on Technology pipeline, High Level Architecture diagram
7	Sprint 1: Learning to use Django, set up basic project infrastructure
8	Sprint 2: Development of 2 key use cases, automated test cases scripting

9	Sprint 3: Another use case, with a focus on Design Patterns
10	Sprint 4: Another use case with more emphasis on Design Patterns, Added Value
11	Architecture and Design Recovery
12	Documentation

4. Requirements

4.1 Functional Requirements

For a Customer

a. Request for bin pickup

i. Customer has the option to raise pickup requests for his/her bins for a date.

ii. The request can be regular or a priority type. A priority request means the request will be completed on the date of customer's choice, while a regular request has a 98% chance of being fulfilled on the date of customer's choice, with 2% chance of being pushed a day or 2 depending on the demand.

b. View past requests

i. Customer should be able to see past requests raised which includes completed and cancelled requests.

c. View & Pay monthly/pay-per-use bills for requests

i. Customer should be able to view the generated bill for the usage and make payments with more than 1 option.

d. Report pickup related issues

i. Customer should be able to report issues with respect of requests, pickup and payments.

e. Edit details

i. Customer should be able to edit basic details like change of address and change of bin numbers (request additional bins for a waste type or reduce)

For Bin Company

a. View demand from each Block (neighborhood) & Route

i. The company should be able to view the demand (list of pickup requests) from a block and route.

b. Use the demand to create bin pickup trips in prefixed route

i. The company should be able to schedule a trip that covers the maximum truck load while at the same time reducing the distance travelled per trip which saves on fuel expenses.

c. Ability to create/delete/modify entities in the system

i. Customer: New customer, edit existing customer, edit bin details

ii. Requests: Create a request for a customer, if the customer is not able to do it in his/her system or change an existing request.

iii. Block (neighborhood): Create or delete a Block as per the demand requirements. Also, to be able to assign/change the block to a route.

iv. Route: Company should be able to create/edit/delete = the predetermined path

v. Truck: Company should be able to add or remove truck based on fleet details

vi. Driver: Company should be able to onboard/deboard a new driver along with changing the existing driver details.

vii. Trips: Company should be able to create pickup trips manually and automatically every day. Also, it should have the ability to cancel a scheduled trip which will notify all the affected customers.

d. Generate bills for customers

i. Use the subscription system and request type to generate bills for the customers based on their usage.

e. Use analytics to predict usage pattern of customers on a route or block

i. Estimate the usage and peak days during the week and festive season to better manage company resources.

For Truck Driver (Bin Collector)

a. View list of trips assigned for the day

i. Driver once logged into the app should be able to see the number of trips, he/she needs to complete before the day ends.

b. View truck assigned for the trip

i. Driver should be able to see the truck number he is assigned to for a trip.

c. View request details for the trip

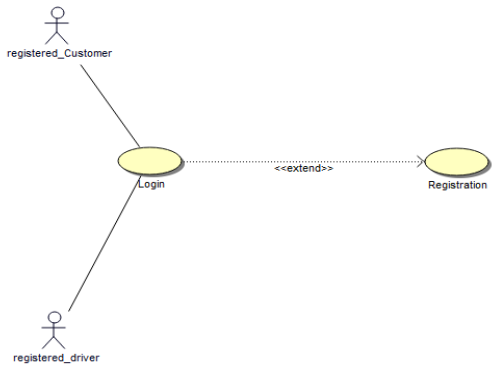
i. Driver should be able to see the pickup request details (number of bins, bin type, address, etc.) for the trip.

d. Report issues for the trip

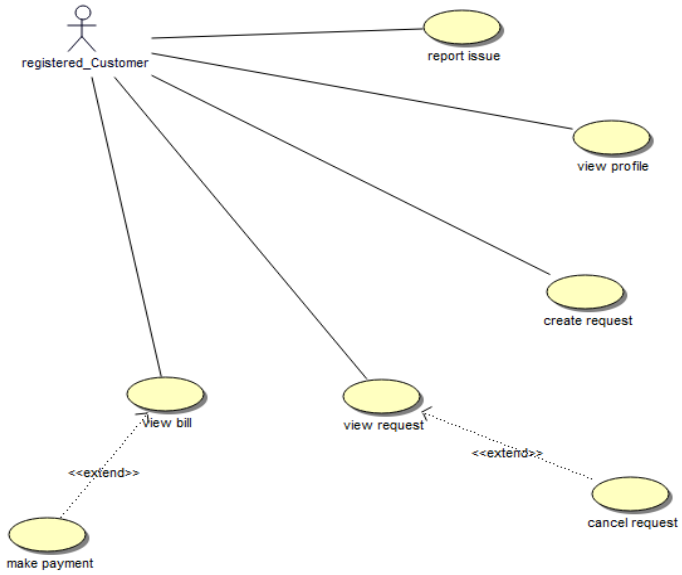
i. Driver should be able to report issues for a trip which may range from bin not present at the address to truck breakdown and traffic blocks.

4.2 Use Case Diagrams

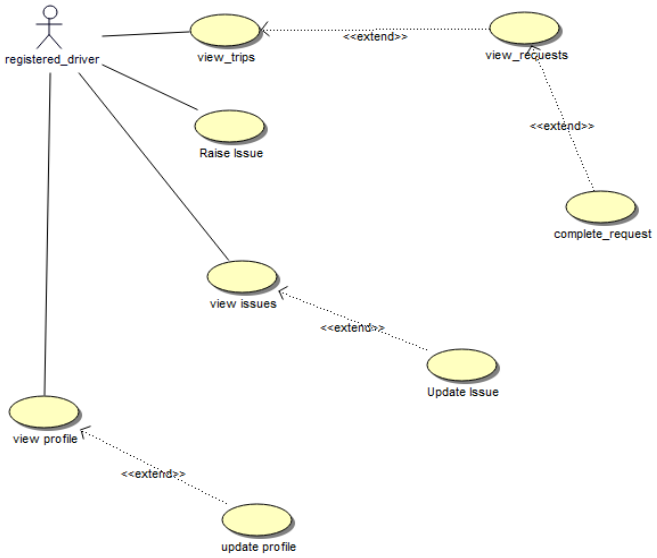
Truck Driver and Customer Log In



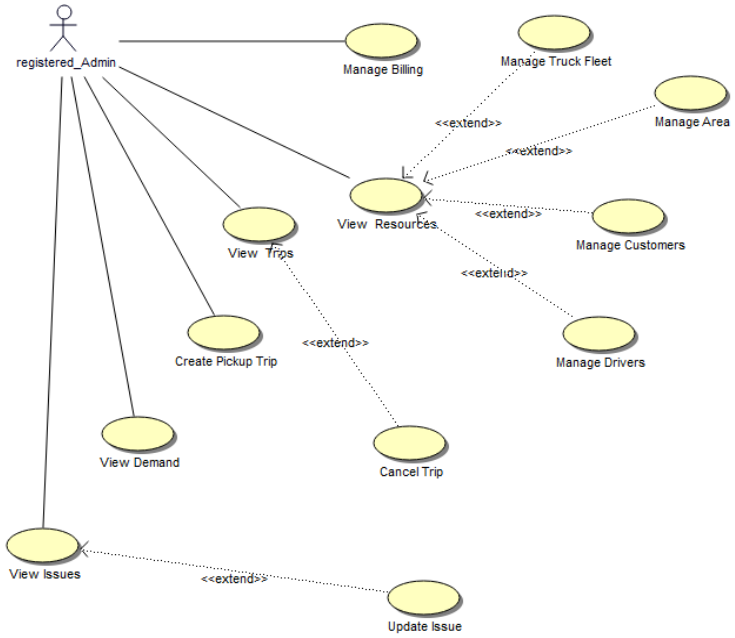
Use Case Diagram for Customer



Use Case Diagram for Truck Driver



Use Case Diagram for Admin



4.3 Use Case Descriptions

Use Case Scenarios for Customer

Use Case	Description
Customer Login	The Customer is asked to login with registered login credentials to create bin collection request.
Customer Registration	If the customer is new, he is asked to create an account and register his details in the system.
Create New Request	After successful login, the customer can create a new request for bin collection by specifying the date of pickup and request type.
View Request	Customer is displayed with both pending requests and completed requests.
Cancel Request	Customer can cancel requests which are scheduled for the pickup.
Report Issue	Customer can report an issue on the way of request handling, bills, payment failure or provide feedback.
View Bill	Customer can see their pending bill payments and past payment history.
Make Payment	Customer pays monthly bill through their preferred mode and gets the status of the payment.

Use Case Scenarios for Truck Driver

Use Case	Description
Driver Login	The driver is authenticated by logging in with registered login credentials to start the trip.
Driver Registration	If the driver is new, he is asked to create an account and register his details in the system.
View Trips	Driver can view all scheduled trips and respective details for the day.
View Requests	After starting a trip, the driver can view all the scheduled requests of the trip that he started.
Complete Request	Once the bin is picked up, the driver can mark the request as complete.
Report Issue	Driver raises an issue in case of pickup failure, truck failure or invalid request details.
Driver - View Issues	Driver is displayed with all the issues and current status of those issues that he logged in the system.
Update Issue	Driver can change, add more information or invalidate the issues he logged.
View Profile	Driver can see his registered personal and contact information in the system.

Update Profile	Driver can update his account details.

Use Case Scenarios for Admin

Use Case	Description
Admin - View Issues	With successful authentication, Admin can see issues raised by driver and customer.
Admin - Update Issue	Admin can update, close or invalidate the issue.
View Demand	Admin is displayed with all pending requests for bin collection and he can sort and filter based on specific area (route).
Create Pickup Trips	Admin can start scheduling new trips for the pending requests.
View Trips	Admin can view trip details like vehicle type, vehicle capacity, driver details, request details.
Cancel Trips	Admin can cancel the scheduled trip which will be notified to both driver and all the customers whose requests are associated to that trip.
Manage Truck Fleet	Admin can add new trucks with valid truck number, size and capacity or edit / delete existing trucks from the system.

Manage Area	Admin can add new area with valid route id and area name or edit / delete existing areas from the system.
Manage Customers	Admin can add / edit / delete customers from the system.
Manage Drivers	Admin can add / edit / delete drivers from the system.

4.3.1 Two Key Use Cases Description

4.3.1.1 Request for waste bin collection

USE CASE 1	Request for waste bin collection
Goal in Context	Customer requests for waste bin collection on a specified date and expects the waste to be collected on the requested date and time.
Scope & Level	Bin Company
Preconditions	Customer should be a registered user with valid address and contact information
Success End Conditions	The waste is disposed of and the company will charge the customer for the waste disposal.

Failed End Condition		The waste is not collected and the customer is not charged for the request.	
Primary, Secondary Actors		Customer (Residence)	
Trigger		Bin collection request comes in	
DESCRIPTION		**Step**	**Action**
		1	Customer logs into the application.
		2	Customer requests for bin collection with a desirable date.
		3	Bin Company notifies the customer with the pickup once the request is scheduled for the pickup.
		4	Customer bin gets collected on the scheduled date.
		5	Customer gets notified on completion of pickup.
		6	Company charges customer after completing the request.

		7	Customer gets bill for all completed requests at the end of the month.
EXTENSIONS		**Step**	**Branching Action**
		2a	Customer has raised request with high priority.
			2a 1. The bin will be picked up on the same day.
			2a 2. Customer will be charged extra for high priority requests.
		3a	Customer is unavailable on the scheduled pickup date.
			3a 1. Cancel the scheduled pickup.
		4a	The customer is notified with pickup completion message, but bin is not collected.
			4a 1. Customer reports an issue.
VARIATIONS			**Branching Action**

		2	Customer may create a request with
			High priority
			Regular

RELATED INFORMATION	**Request for waste bin collection**
Priority:	Top
Performance	Less than a minute for request creation, < 3 days for pick up.
Frequency	Varies based on demand.
Channel to actors	Not yet determined.
OPEN ISSUES	What if the request is rescheduled?

Due Date	Release 1.0
…any other management information…	
Superordinates	Authentication (Log In)
Subordinates	Create Request
	Pay bill by choice of payment mode.

4.3.1.2 Complete customer request / Collect waste for recycling

USE CASE 2	**Complete customer request / Collect waste for recycling**
Goal in Context	Collect waste from requested customer locations in the route.
Scope & Level	Driver, Bin Company
Preconditions	Driver should be registered with the company. (employee)

Success End Conditions		The waste gets collected and driver will be able to complete the trip.	
Failed End Condition		The waste is not collected and the customer's request cannot be completed.	
Primary, Secondary Actors		Truck Driver	
Trigger		Driver starts his assigned trip.	
DESCRIPTION		**Step**	**Action**
		1	Driver logs in to the application.
		2	Drivers goes to his assigned trips page and starts his trip.
		3	Driver views customer request and collects waste bin from the given location.
		4	Driver collects bin on the scheduled date and time.
		5	Driver marks the request as complete.

		6	Driver sees the next request detail on the trip.
		7	Repeat steps 3 to 6.
		8	Driver completes all requests assigned to the trip.
		9	Driver completes the trip.
EXTENSIONS		**Step**	**Branching Action**
		2a	Driver finds the location of address to be invalid.
			2a 1. Reports an issue against the customer request and mark it as not picked.
			2a 2. The state of the request will be changed to pending
		2b	Driver tries to start another assigned trip.
			2b 1. Driver gets an error message “Please complete the trip in progress to start a new trip”.

		6a	Driver cancels the previous trip.
			3a 1. Status of all pending requests on the cancelled trip are changed and the customers will be notified on pickup cancellation.
VARIATIONS			**Branching Action**

RELATED INFORMATION	**Complete customer request / Collect waste for recycling**
Priority:	Top
Performance	Less than a minute for starting and completing the trip in application. The time for trip completion may vary based on the route and number of requests.
Frequency	Varies based on demand.
Channel to actors	Not yet determined.

OPEN ISSUES	
Due Date	Release 1.0
…any other management information…	
Superordinates	Authentication (Log In)
Subordinates	Start trip
	Complete all pending requests assigned to the trip.

4.4 Non-Functional Requirements

Scalability

As the business grows, there will be pressure on the system resources to provide low latency access to more and more customers simultaneously. Once the upper limit is reached within the existing system, there will be only 2 options:

1. Scale the system horizontally
2. Scale the system vertically

The design of Django enables us to scale while keeping the different layers independent of each other. Application servers being stateless are scalable horizontally. However, since the database is primarily responsible for the scale state of the Django system, we might need to introduce database caching and sharding in future to maintain low latency of the database.

Django application also has 3 types of caching - Per-site caching, Per-view caching and template fragment caching. A combination of these are used to reduce the application loading time per user. Additionally, the use of Nginx web server and load balancer as an intermediate layer between the web client and wsgi server helps us maintain the separation of layers and scale the presentation and the application layer independently as per the requirements.

Availability

There are serious limitations on having an application hosted in-house. The need for dedicated personnel and resources to keep the application running 24x7 seriously inhibits company's time and finances. The solution is to have the application hosted on cloud and transfer the responsibility to a cloud service provider.

The service should have an extremely low probability of failure (<0.001%) and possess redundancy options in cases of fail-overs. This provides a way for the company to have guaranteed uptime and very less chances of loss of business due to application or hardware crashes. After careful examination we feel that the Heroku Cloud Application Platform is the best option to deploy Django Application due to its native support of Python applications and MySQL databases.

Extensibility

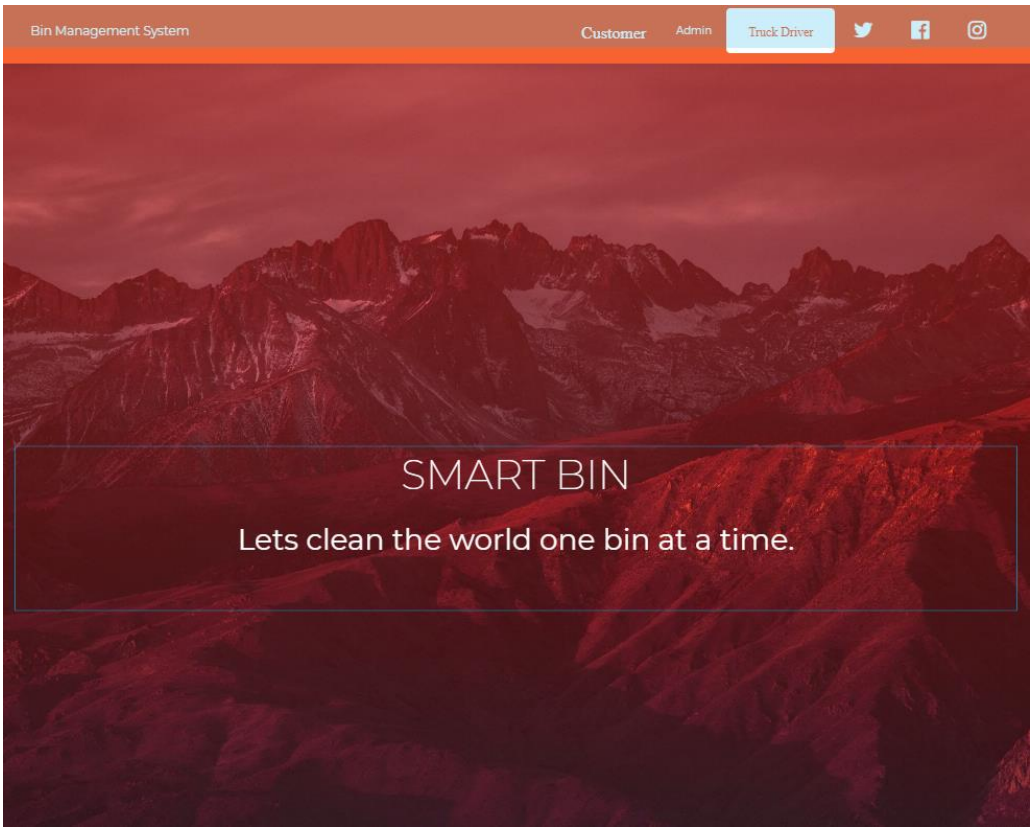
Once the initial application is stable, there will come a need to add new functionalities from the primary 3 actors in our case - Customer, Company (Admin) and the Truck Driver (Bin collector).

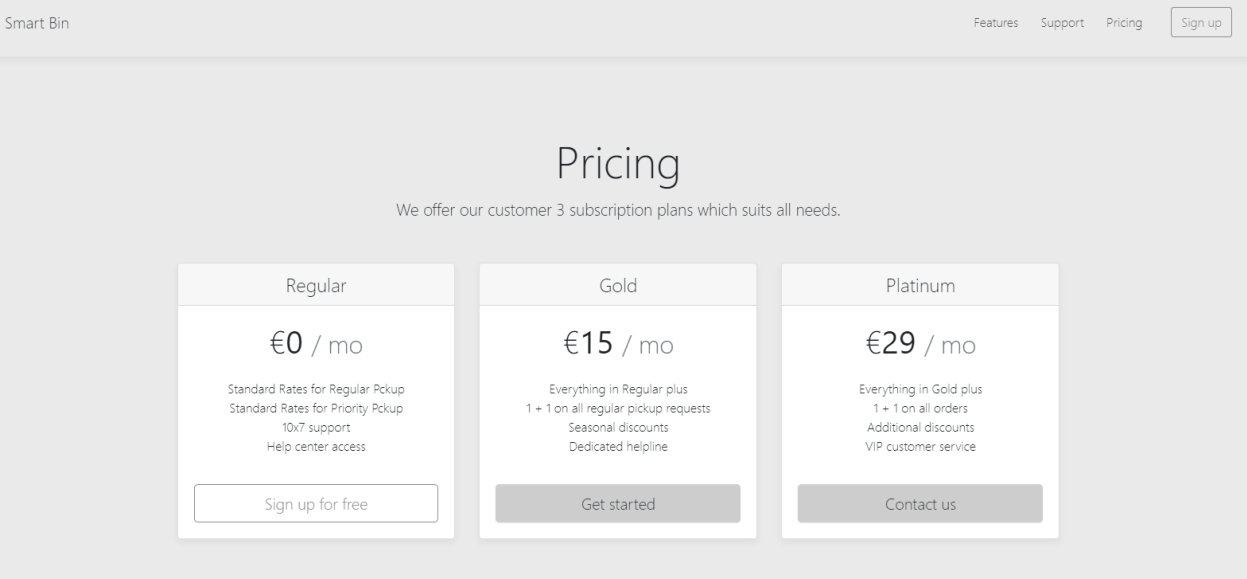
Due to this reason, the application is developed in a very modular way where each set of related functionalities lie in its own Django app (package). This separation allows us to independently work on the Django app without affecting any unrelated packages. Also, Django allows us to create any number of apps within the project and add them using install_app feature. We can also remove an app from the project by removing it from the installed apps. This keeps each app decoupled from the other and modularizes the whole project.

Serviceability

There may also be a need to modify existing functionalities based on the feedback from the actors. As stated above, the Django app-based segregation provides us the functionality to make modification on an existing app without affecting the other components.

4.5 GUI Prototypes





5. Architectural Patterns

5.1 Model-View-Controller (MVC)

Model-View-Controller (usually known as MVC) is a software architectural pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This is mainly done to separate internal representations of information from the ways information is presented to and accepted from the user. Following the MVC architectural pattern decouples these major components allowing for code reuse and parallel development. Traditionally used for desktop graphical user interfaces (GUIs), this pattern has become popular for designing web applications. Popular programming languages like JavaScript, Python, Ruby, PHP, Java, and C# have MVC frameworks that are used in web application development straight out of the box.

Model
The central component of the pattern. It is the application's dynamic data structure, independent of the user interface. It directly manages the data, logic and rules of the application. The model is responsible for managing the data of the application. It receives user input from the controller.

View
Any representation of information such as a chart, diagram or table. Multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The view means presentation of the model in a particular format.

Controller
Accepts input and converts it to commands for the model or view. The controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

As with other software patterns, MVC expresses the "core of the solution" to a problem while allowing it to be adapted for each system.

5.2 N-Tier Architecture

In software engineering, multitier architecture (often referred to as n-tier architecture) or multilayered architecture is a client-server architecture in which presentation, application processing, and data management functions are physically separated. N-tier application architecture provides a model by which developers can create flexible and reusable applications. By segregating an application into tiers, developers acquire the option of modifying or adding a specific layer, instead of reworking the entire application.

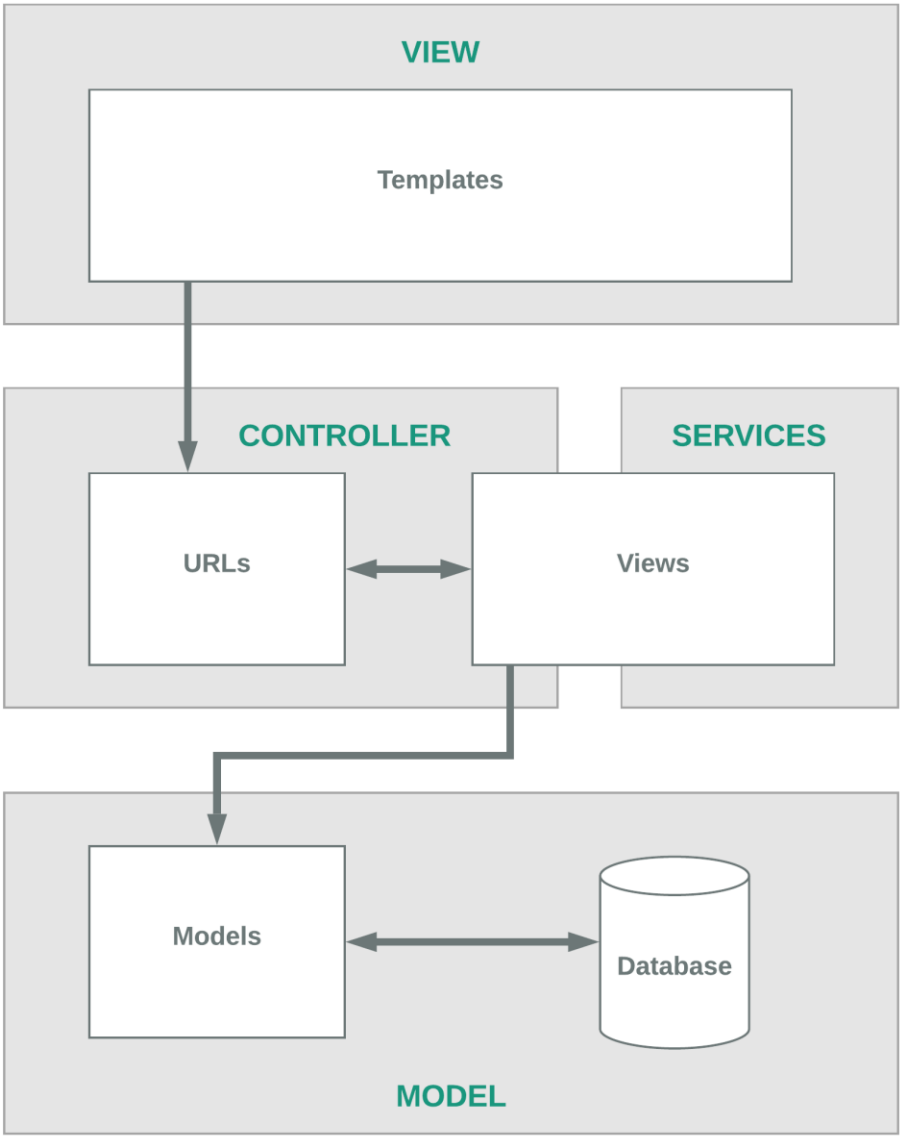
The separate physical location of these tiers is what differentiates n-tier architecture from the model-view-controller architecture framework that only separates presentation, logic, and data tiers in concept. N-tier architecture also differs from MVC framework in that the former has a middle layer or a logic tier, which facilitates all communications between the different tiers. On using the MVC framework, the interaction that happens is triangular; instead of going through the logic tier, it is the control layer that accesses the model and view layers, while the model layer accesses the view layer. Additionally, the control layer makes a model using the requirements and then pushes that model into the view layer.

Benefits of employing N-tier architecture include:

More efficient development. N-tier architecture is very friendly for development, as different teams may work on each tier. This way, one can be sure that the design and presentation professionals work on the presentation tier and the database experts work on the data tier.
Easy to add new features. New features can be added to the appropriate tier without affecting the other tiers.
Easy to reuse. Because the application is divided into independent tiers, it is possible to reuse each tier for other software projects. For instance, using the same program, but for a different data set, one can just replicate the logic and presentation tiers and then create a new data tier.

6. System Architecture

6.1 High Level Architecture Diagram



6.2 Technology Pipeline

Django

Django is a Python-based free and open-source web framework, which follows the model-template-view (MTV) architectural pattern. It is maintained by the Django Software Foundation (DSF), an independent organization established as a non-profit. Django's primary goal is to ease the creation of complex, database-driven websites. The framework emphasizes reusability and "pluggability" of components, less code, low coupling, rapid development, and the principle of “don't repeat yourself”. Python is used throughout, even for settings files and data models. Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models.

MySQL

MySQL is an open-source relational database management system (RDBMS). MySQL is a component of the LAMP web application software stack (and others), which is an acronym for Linux, Apache, MySQL, Perl/PHP/Python. It is widely used by many database-driven web applications and has been tested to be a fast, stable and true multi-user, multi-threaded SQL database server. Though MySQL began as a low-end alternative to more powerful proprietary databases, it has gradually evolved to support higher-scale needs as well. Much of MySQL's appeal originates in its relative simplicity and ease of use, which is enabled by an ecosystem of open source tools such as phpMyAdmin. In the medium range, MySQL can be scaled by deploying it on more powerful hardware, such as a multi-processor server with gigabytes of memory.

Bootstrap

Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains CSS and JavaScript-based design templates for typography, forms, buttons, navigation and other interface components. It is a web framework that focuses on simplifying the development of informative web pages. The primary purpose of adding it to a web project is to apply Bootstrap's choices of color, size, font and layout to that project. Once added to a project, Bootstrap provides basic style definitions for all HTML elements. The result is a uniform appearance for prose, tables and form elements across web browsers. In addition, developers can take advantage of CSS classes defined in Bootstrap to further customize the appearance of their contents. Bootstrap also comes with several JavaScript components in the form of jQuery plugins. They provide additional user interface elements such as dialog boxes, tooltips, and carousels.

Nginx

Nginx (pronounced "engine X") is a web server which can also be used as a reverse proxy, load balancer, mail proxy and HTTP cache. The software was first publicly released in 2004. Nginx is free and open-source software, released under the terms of a BSD-like license. A large fraction of web servers use NGINX,

often as a load balancer. Nginx can be deployed to serve dynamic HTTP content on the network using FastCGI, SCGI handlers for scripts, WSGI application servers or Phusion Passenger modules, and it can serve as a software load balancer. Nginx uses an asynchronous event-driven approach, rather than threads, to handle requests. Nginx's modular event-driven architecture can provide more predictable performance under high loads.

CircleCI

CircleCI's continuous integration and delivery platform helps software teams rapidly release code with confidence by automating the build, test, and deploy process. CircleCI offers a modern software development platform that lets teams ramp quickly, scale easily, and build confidently every day. CircleCI enables developers to detect and fix bugs before they even reach customers. Thousands of leading companies including Facebook, Kickstarter, Shyp, and Spotify rely on CircleCI to accelerate the delivery of their code and enable developers to focus on creating business value fast.

7. Analysis artefacts

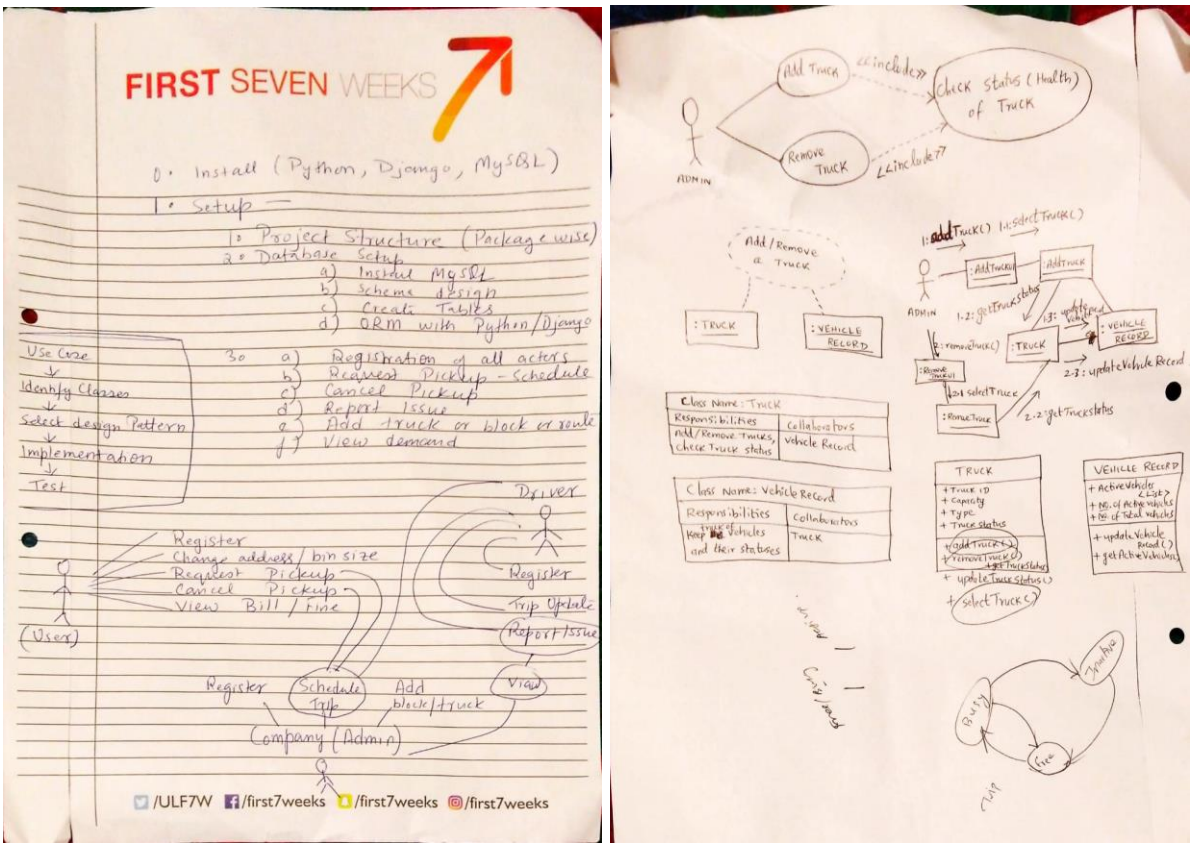
7.1 Use Case Realization

Use Case Realization is an excellent form of requirements traceability from the logical business requirements down to the physical implementation of the solution. According to RUP, Use Case Realization creates the Analysis Model consisting of Boundary classes, Control classes and Entity classes in Class Diagrams, Sequence Diagrams. These Models are logical models describing Whitebox inner workings of the system (as opposed to the use cases that treat the system as a Blackbox). Use Cases are the WHAT, the functional requirements. Use Case Realizations are the HOW, the solution (at the logical level).

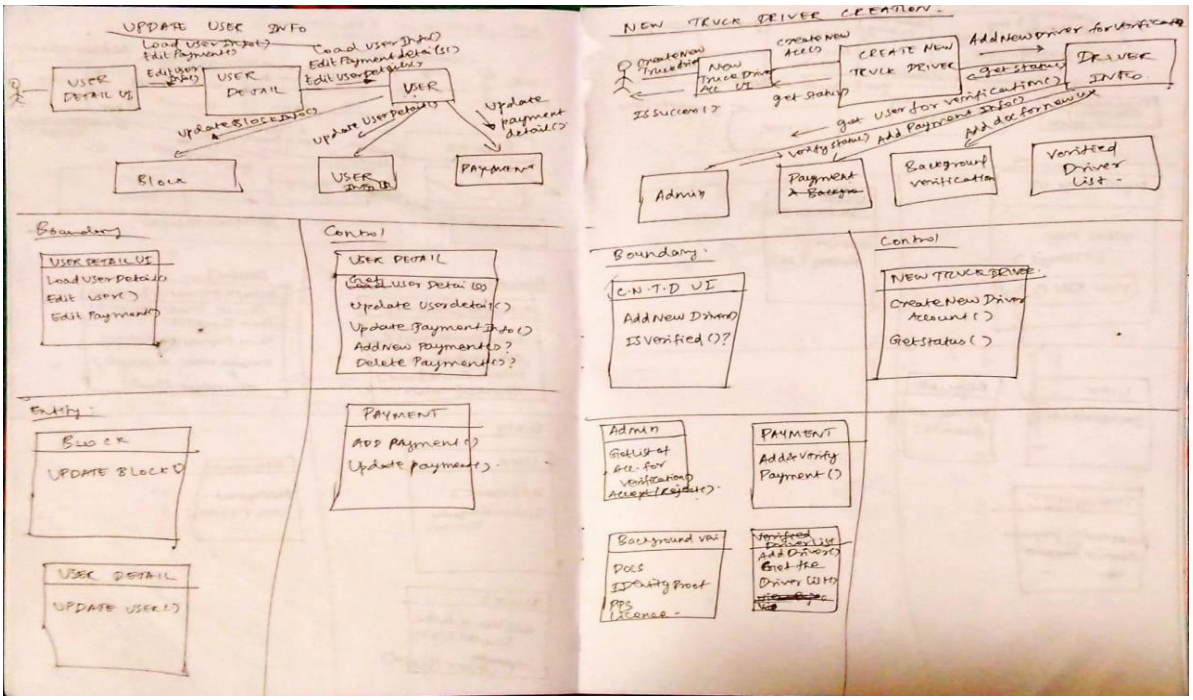
We performed Use Case Realization and arrived at the Analysis Class Diagram by following the below steps in sequence:

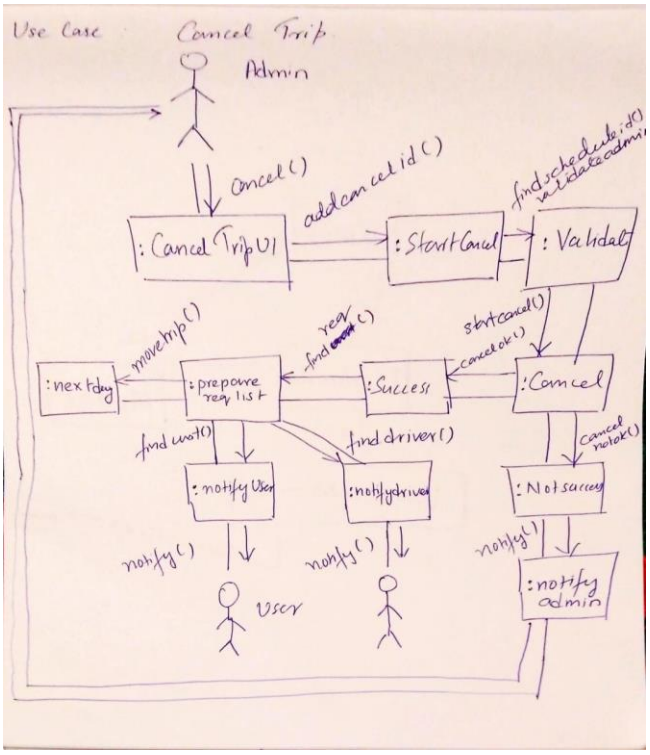
For each use case,

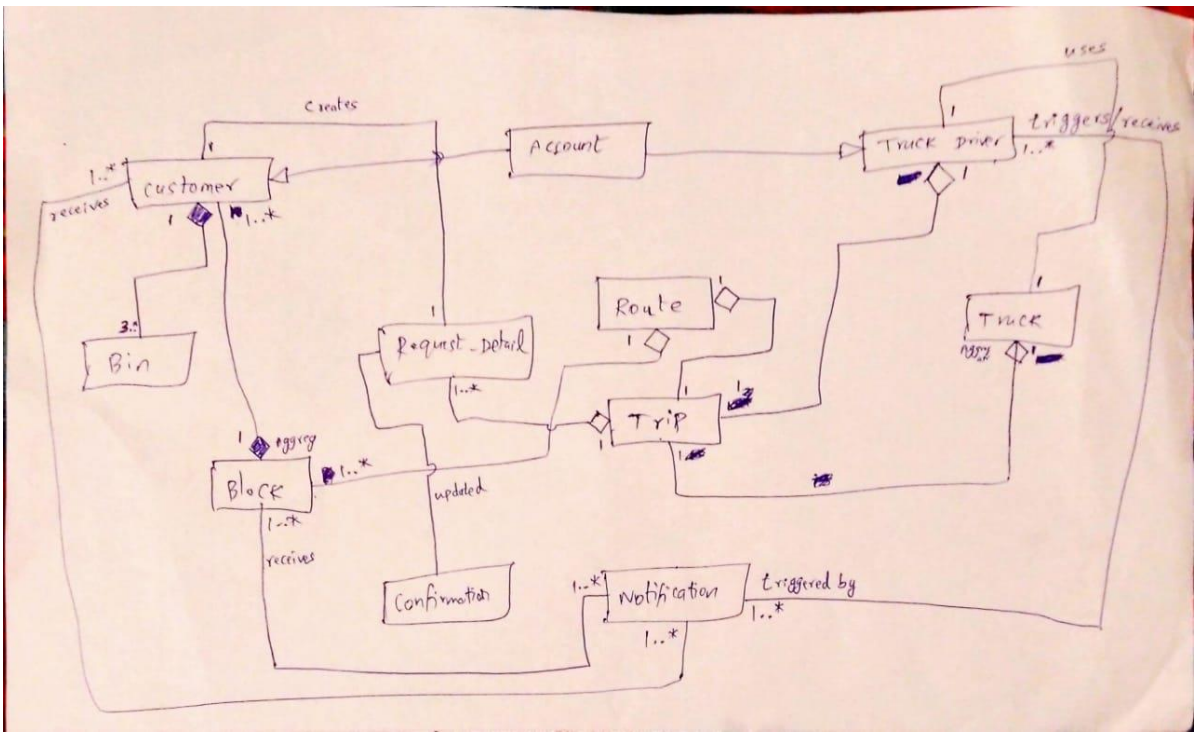
- List potential candidate classes using noun-identification technique
- Discard inappropriate classes using heuristics
- Draw a collaboration diagram, communication diagram and use Class Responsibilities Collaborations (CRC) cards to sanity check interactions
- Separate the classes into boundary, control and entity classes
- The fragment of class diagram derived from this use case realization is called use case class diagram
- Integrate all these use case class diagrams into one and sanity check the associations
- Identify generalizations, add aggregation and composition


FIRST SEVEN WEEKS
0. Install (Python, Django, MySQL)
1. Setup —
1. Project Structure (Package wise)
2. Database Setup
a) Install MySQL
b) Schema design
c) Create Tables
d) ORM with Python/Django
3. a) Registration of all actors
b) Request Pickup - Schedule
c) Cancel Pickup
d) Report Issue
e) Add truck or block or route
f) View demand
Use Case
Identify Classes
Select design Pattern
Implementation
Test
Register
Change address / bin size
Request Pickup
Cancel Pickup
View Bill / Fine
(User)
Driver
Register
Trip Update
Report Issue
View
Register
Schedule Trip
Add block/truck
Company (Admin)
/ULF7W /first7weeks /first7weeks /first7weeks

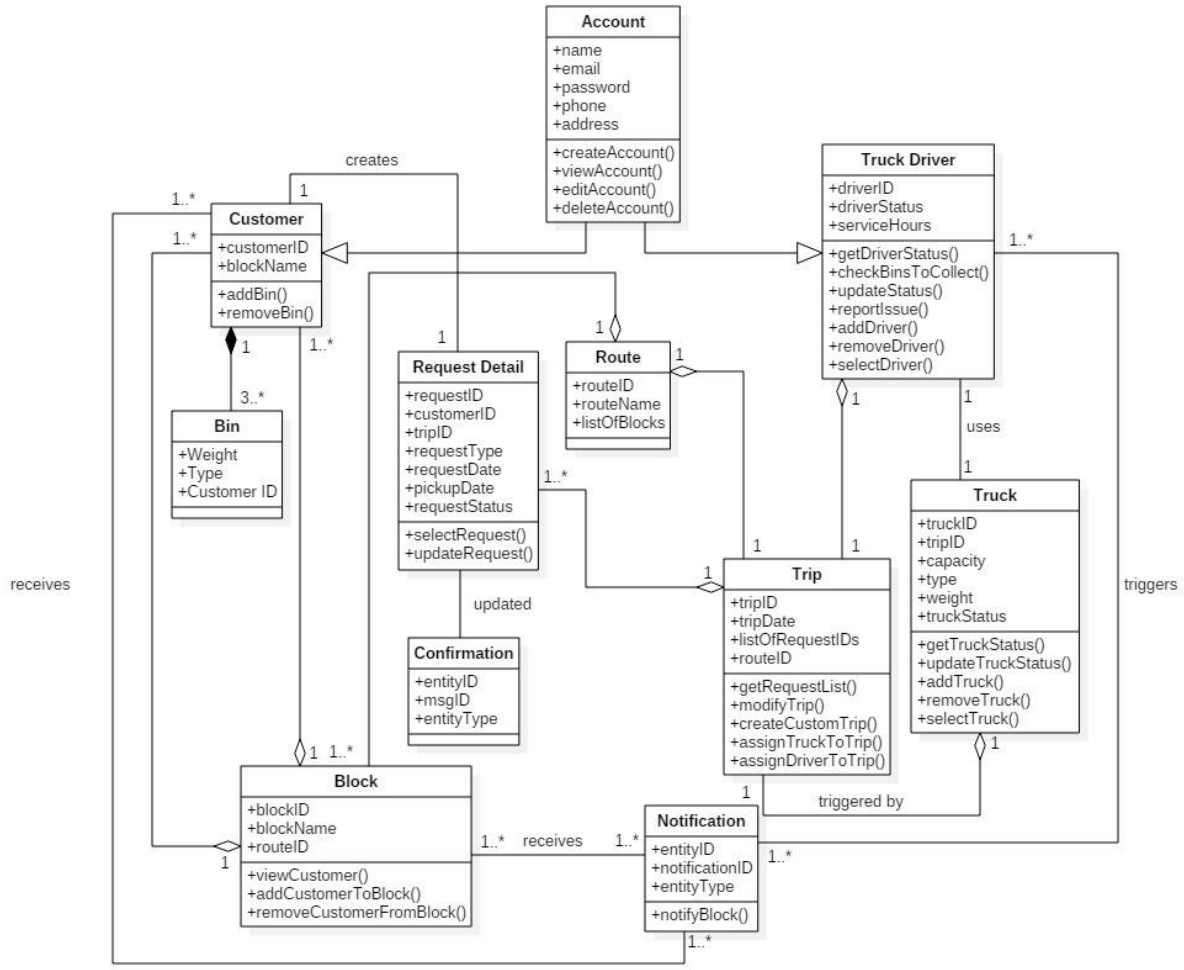
Add Truck
«include»
Check Status (Health) of Truck
ADMIN
Remove Truck
«include»
Add/Remove a Truck
: TRUCK
: VEHICLE RECORD
ADMIN
: AddTruck
: RemoveTruck
Class Name: TRUCK
Responsibilities
Collaborators
Add/Remove Trucks, check Truck status
Vehicle Record
Class Name: Vehicle Record
Responsibilities
Collaborators
Keep the Vehicles and their statuses
TRUCK
+ Truck ID
+ Capacity
+ Type
+ Truck status
+ addTruck()
+ removeTruck()
+ getTruckStatus()
+ updateTruckStatus()
+ selectTruck()
VEHICLE RECORD
+ Active Vehicle List
+ No. of Active vehicles
+ No of Total vehicles
+ updateVehicleRecord()
+ getActiveVehicles()
Active
Busy
Idle
Out of Service


UPDATE USER INFO
USER DETAIL UI
USER DETAIL
USER
BLOCK
USER
PAYMENT
Boundary
USER DETAIL UI
Load User Details()
Edit User()
Edit Payment()
Control
USER DETAIL
Get User Details()
Update UserDetails()
Update Payment Info()
Add New Payment()?
Delete Payment()?
Entity
BLOCK
UPDATE BLOCK()
USER DETAIL
UPDATE USER()
PAYMENT
Add Payment()
Update payment()
NEW TRUCK DRIVER CREATION
Admin
Payment
Background verification
Verified Driver List
Boundary
C.N.T.D UI
Add New Driver()
Is Verified()?
Control
NEW TRUCK DRIVER
Create New Driver Account()
Get Status()
Admin
Get List of Acc. for verification
Accept/Reject()
PAYMENT
Add & Verify Payment()
Background Ver
Docs
ID Verify Proof
PPS
License
Verified Driver List
Add Driver()
Get the Driver List()

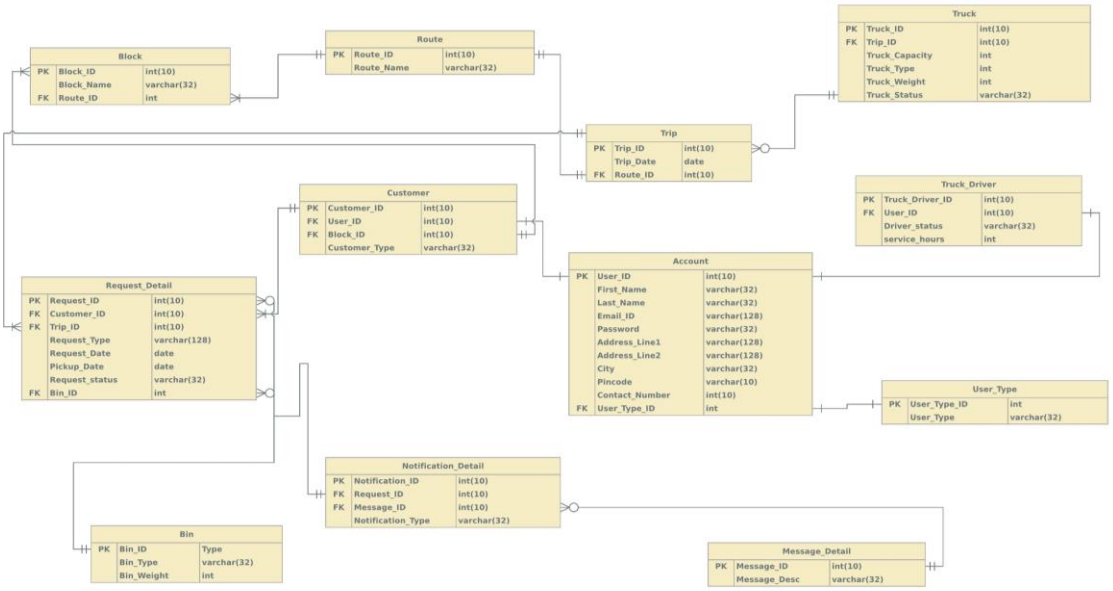

Use Case
Cancel Trip
Admin
cancel()
addcancelid()
: CancelTripUI
: StartCancel
: Validate
: nextday
moveTrip()
: prepareReqList
: Success
: Cancel
: notifyUser
: notifydriver
: Notsuccess
: notify admin
User


creates
Account
uses
Truck Driver
triggers/receives
customer
receives
Bin
Request_Detail
Route
Truck
Trip
Block
updated
Confirmation
Notification
triggered by

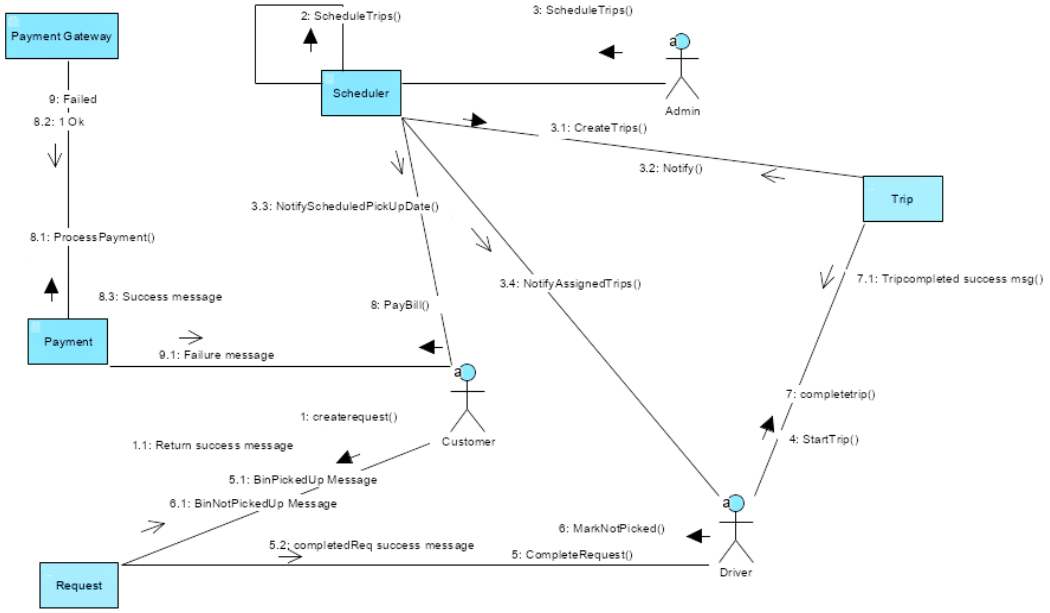
7.2 Analysis Class Diagram



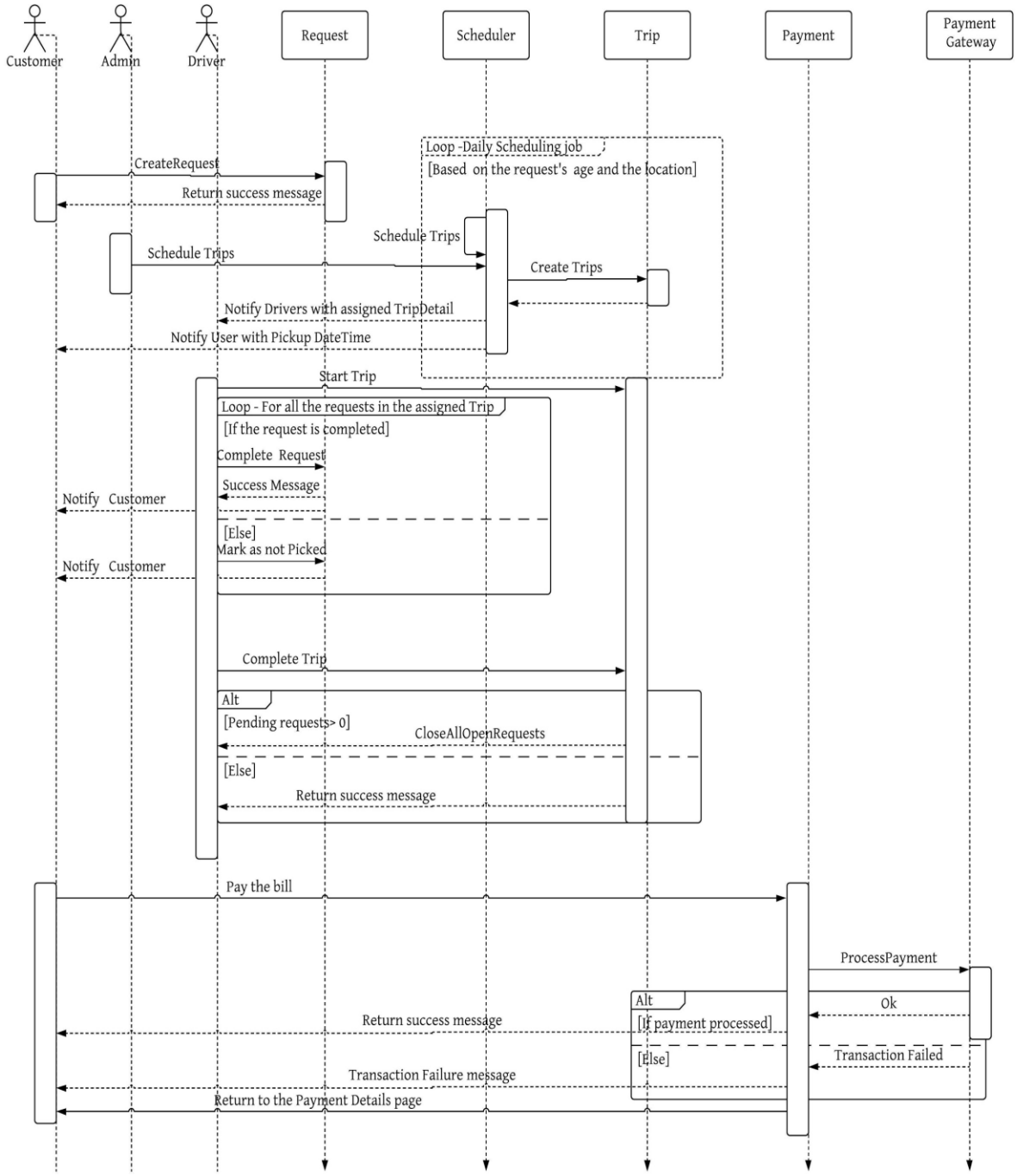
7.3 Entity Relationship Diagram



7.4 Communication Diagram

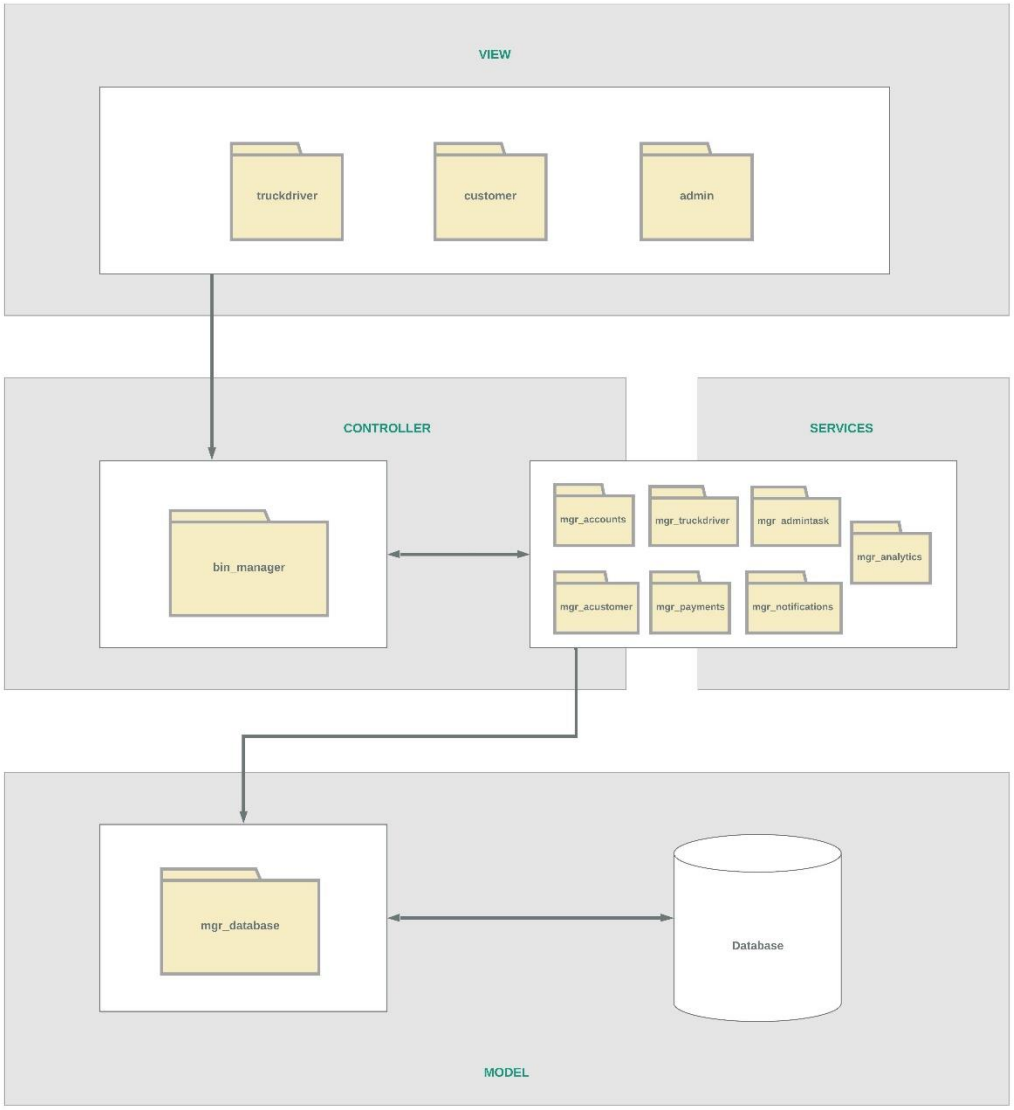


7.5 Sequence Diagram

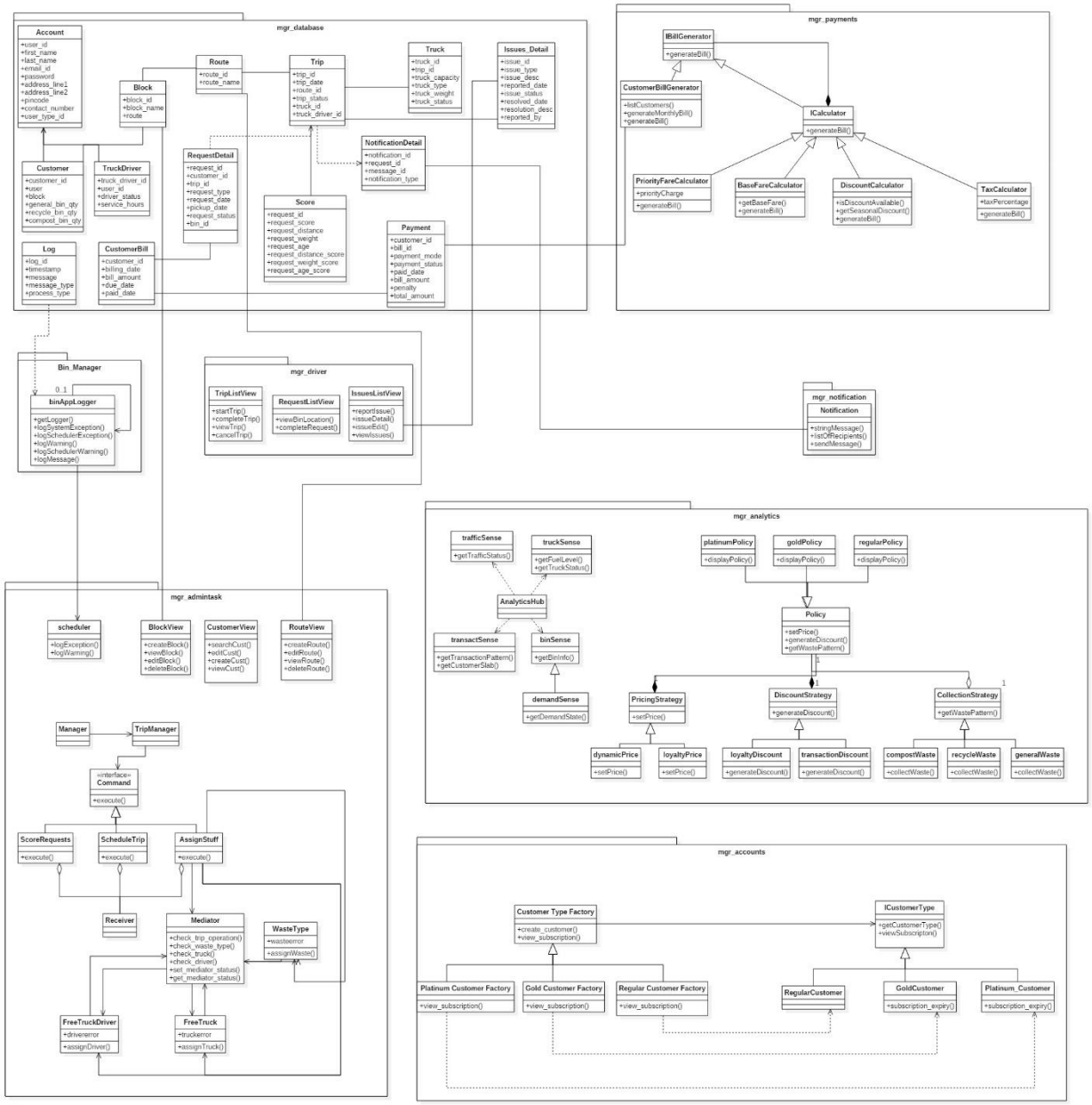


8. Recovered Blueprints

8.1 Recovered Architectural Hierarchy/ Package Diagram

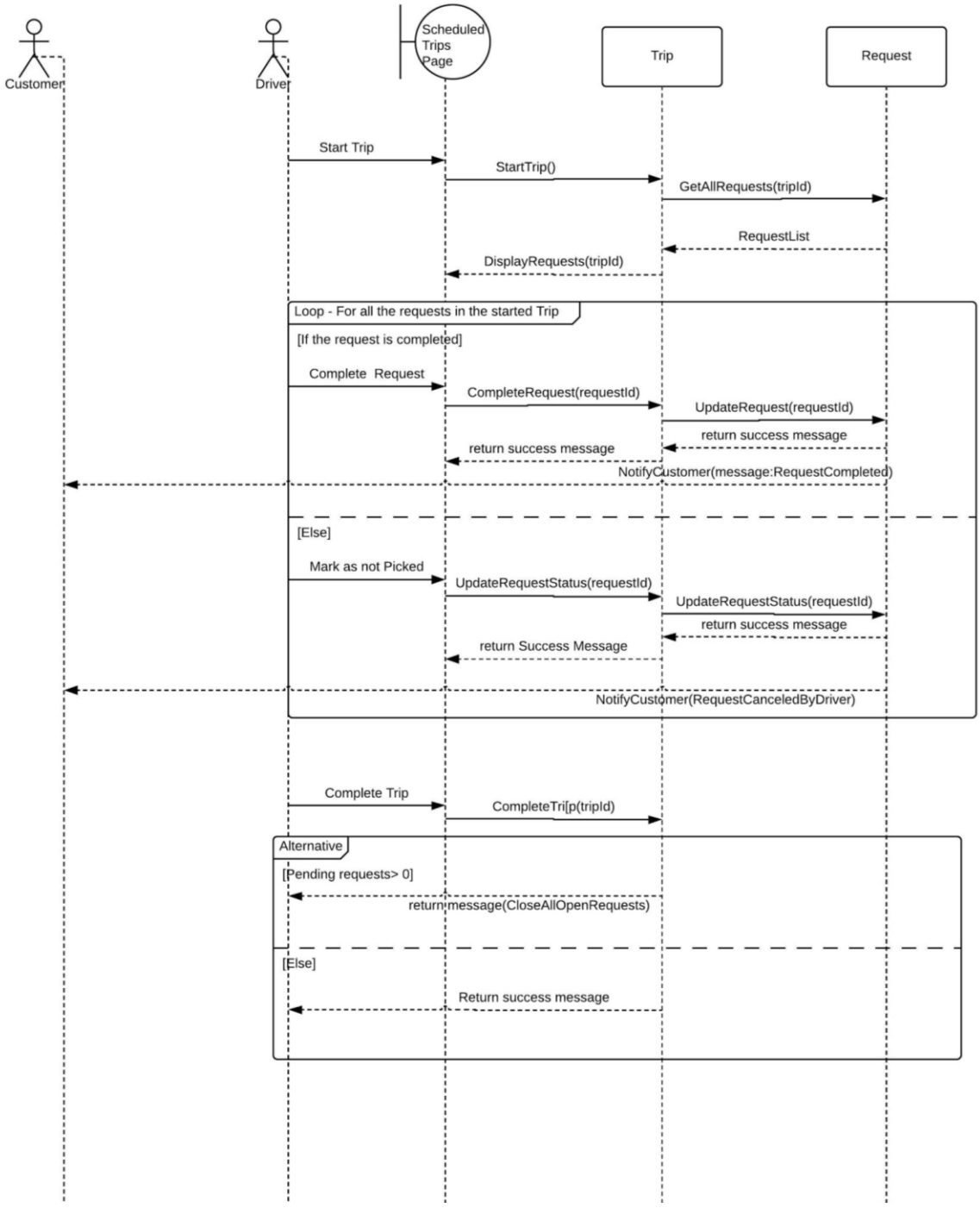


8.2 Design Time Class Diagram

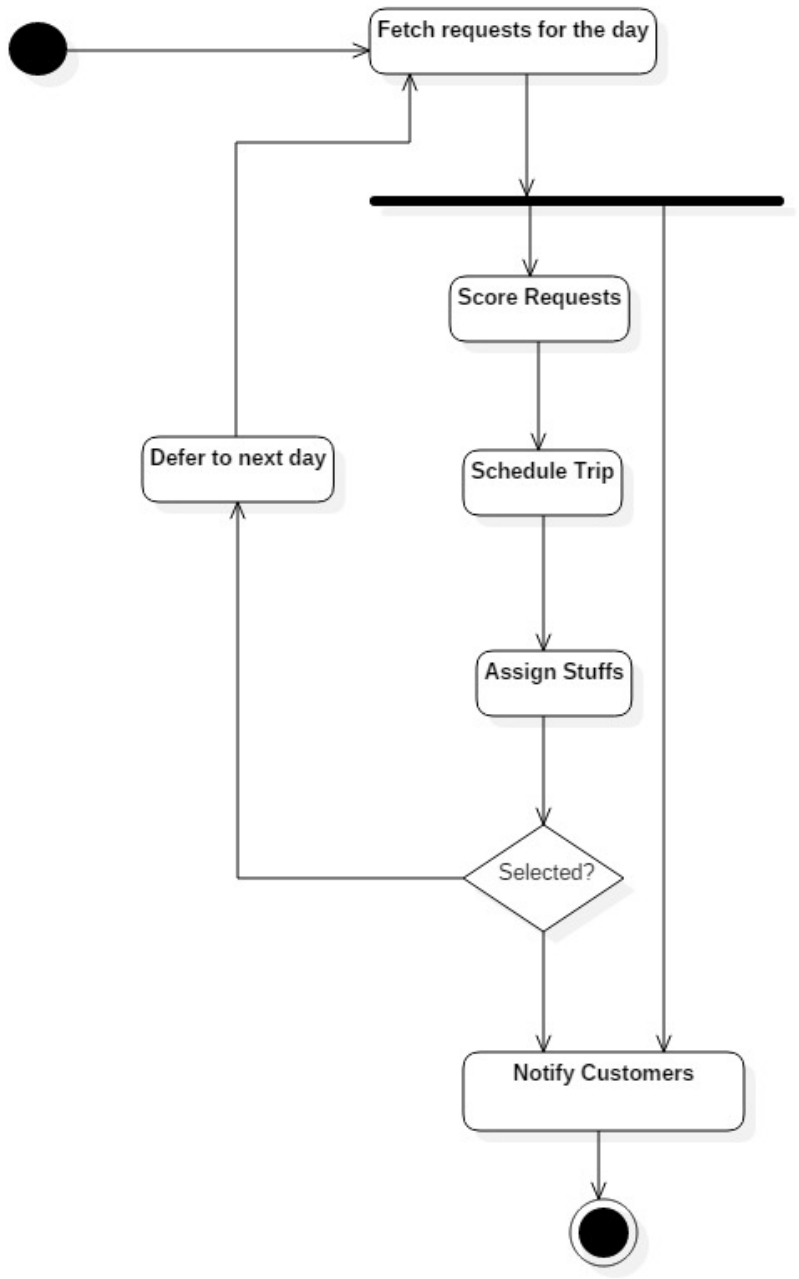


8.3 Design Time Interaction Diagram

Complete customer request / Collect waste for recycling

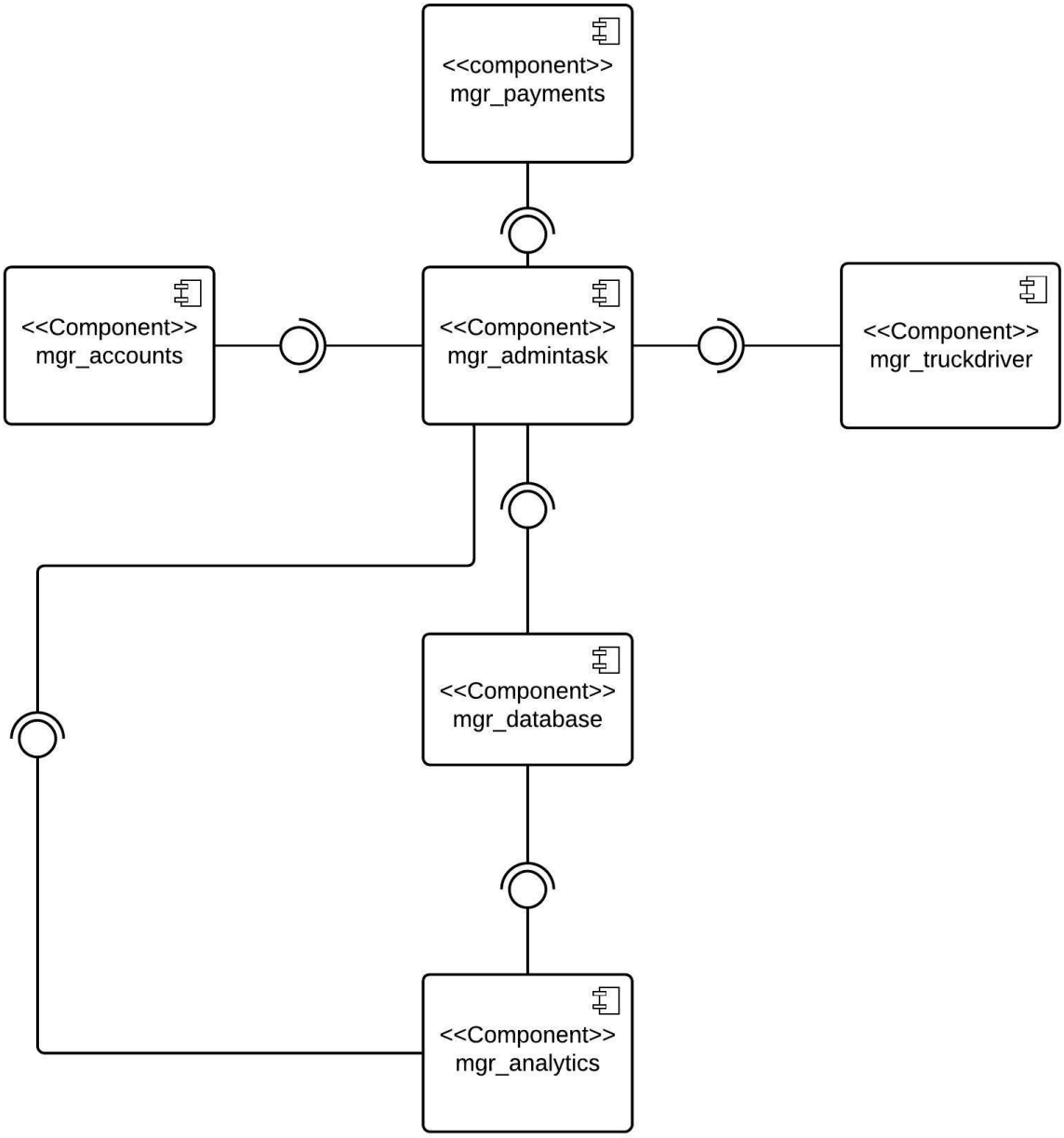


8.4 State Chart Diagram

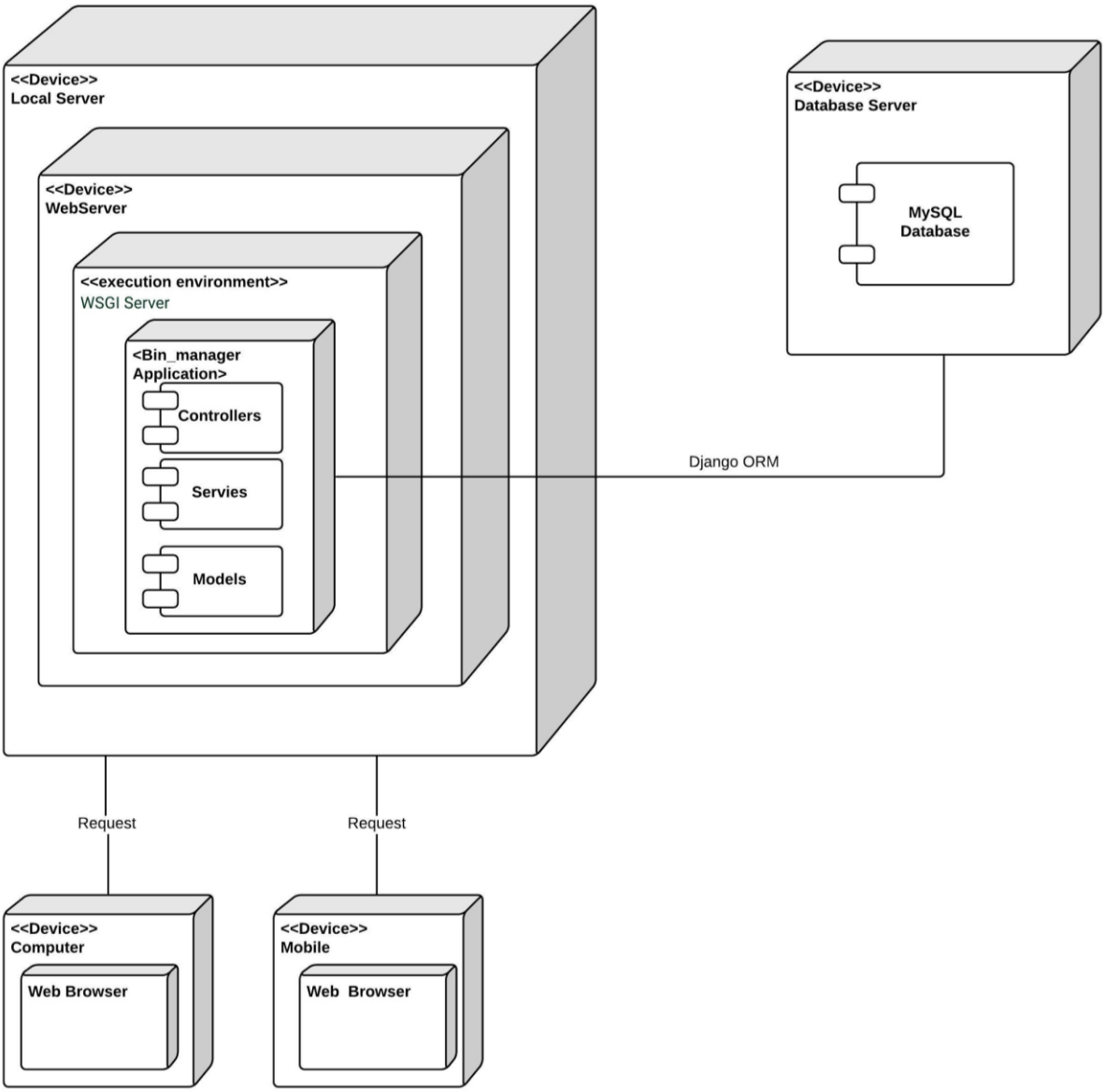


9. Component & Deployment Diagrams

9.1 Component Diagram



9.2 Deployment Diagram



10. Design Patterns

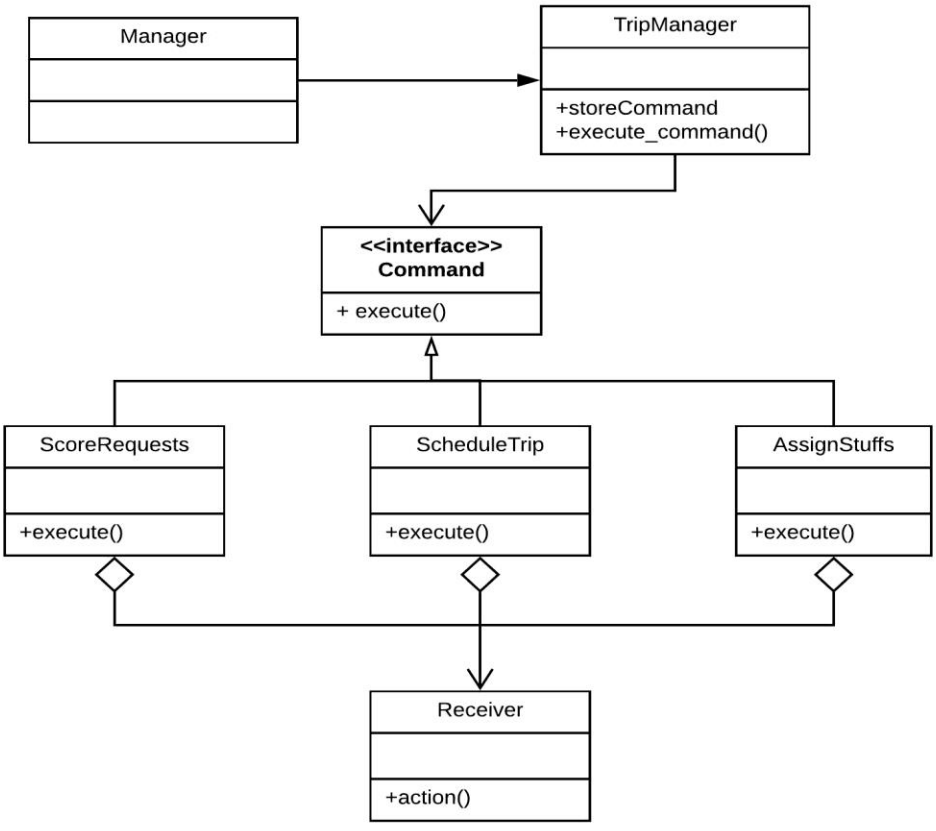
10.1 Command Pattern

Definition

The Command pattern encapsulates a request in an object, which enables you to store the command, pass the command to a method, and return the command like any other object.

Implementation

The command pattern is employed as the base design pattern for running the 3-stage process of creating a trip. We encapsulate 3 commands of scoring, scheduling and assigning in 3 objects derived from an abstract class and invoke execute method of each object to complete the task. The receiver class is used to keep track of the stages that are being completed.



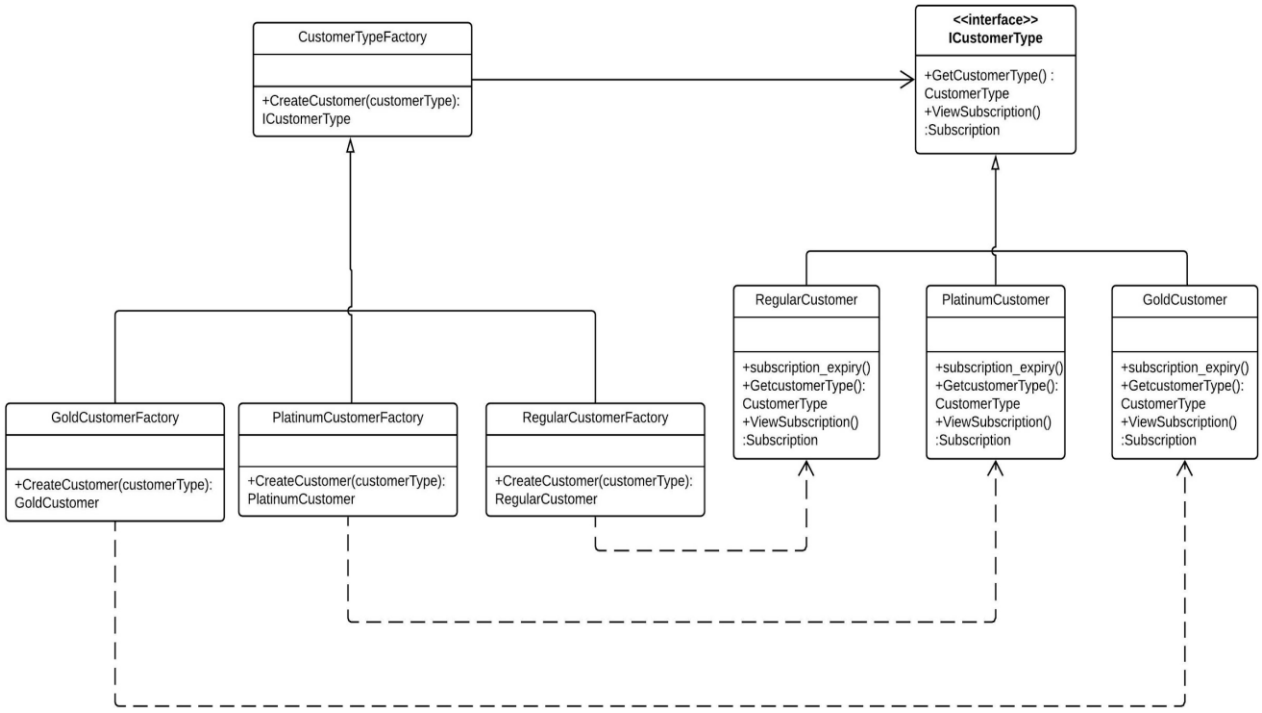
10.2 Factory Method Pattern

Definition

The factory method defines an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Implementation

The application uses CustomerTypeFactory to create different types of customers. It is used to replace class constructors, abstracting the process of object generation so that the type of object instantiated can be determined at run-time.



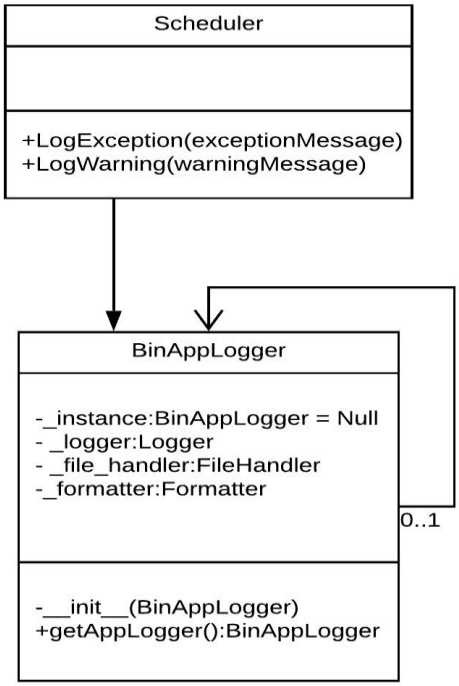
10.3 Singleton Pattern

Definition

The Singleton pattern is one of the creational patterns which helps to ensure that a class has only one instance and provides a global point of access to it. The key idea in this pattern is to make the class itself responsible for controlling its instantiation (that it is instantiated only once) and also to have controlled access to the instance.

Implementation

In our application, we have made our BinApplogger as singleton class which serves as a central logging system. The instance of this class will be created when the application starts and will be preserved till the application is stopped by the webserver. All the webapps and background services (Scheduler and BillGenerator) will log the information through this single instance of BinAppLogger.



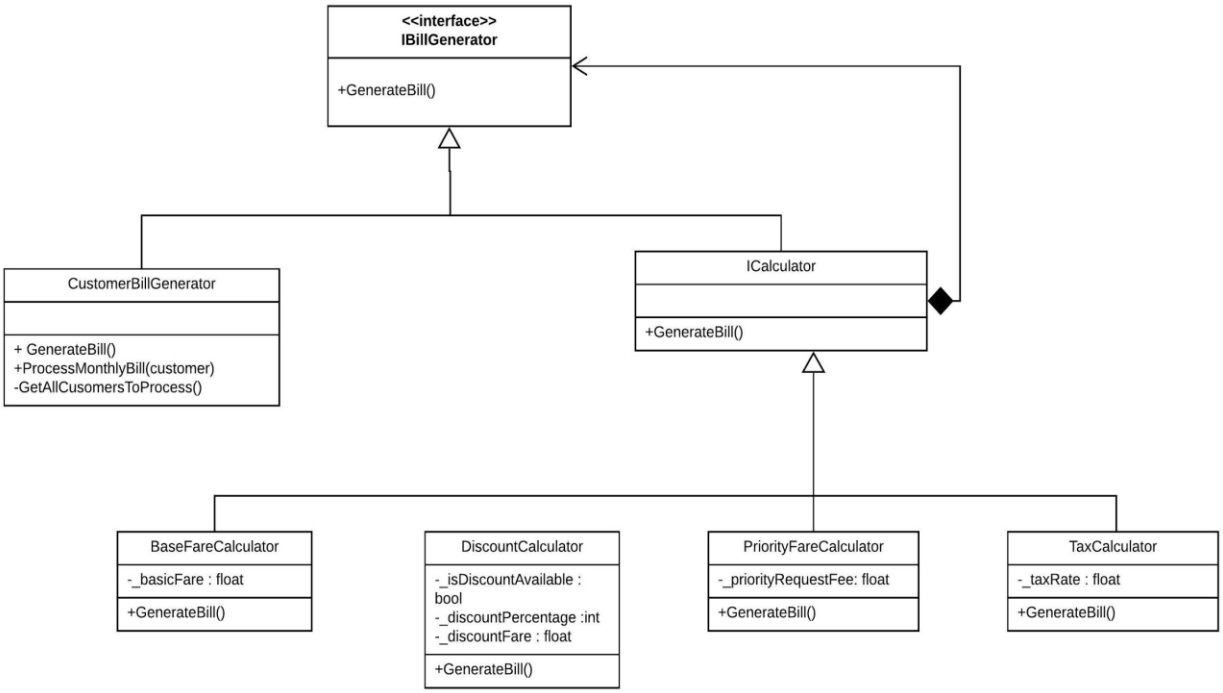
10.4 Decorator Pattern

Definition

The decorator design pattern is one of the GOF design patterns that serves the purpose of adding responsibility to an object dynamically. This helps to adhere to the Single Responsibility Principle, as it allows functionality to be divided between classes and also serves as an alternative for subclassing.

Implementation

We have used the decorator pattern to implement monthly bill generation for different types of customers (Regular, Gold, Platinum). The Process calls CustomerBillGenerator through IBillGenerator interface. The GenerateBill method adds additional charges like discount, priority charges and tax to the base fare dynamically based on the customerType.



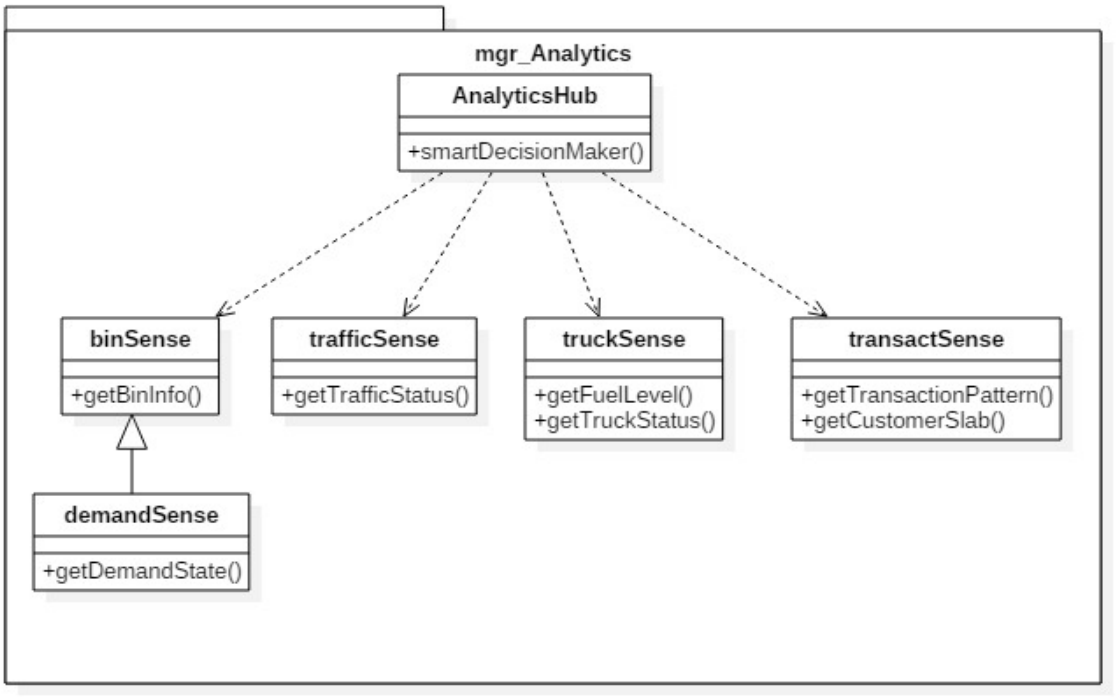
10.5 Facade Pattern

Definition

The Facade pattern provides a unified interface to a group of interfaces in a subsystem. The Facade pattern defines a higher-level interface that makes the subsystem easier to use because you have only one interface. This unified interface enables an object to access the subsystem using the interface to communicate with the subsystem.

Implementation

The bin management system has several sub-systems such as customer demand sense, customer transaction pattern analyzer, traffic sensing and real time truck sensing. We believe that, using these subsystems can help in significant cost savings for the organization. However, making decisions by accessing each of these single subsystems would be a complex task. To simplify this, we propose the use of the 'Facade' design pattern which is a Structural Design Pattern. In our system, the class 'AnalyticsHub' acts as a facade, providing a single interface to the underlying subsystems through a single method. This abstracts the underlying subsystems' implementation to the client, and only exposes itself as a method to the client, enabling an easier interface to the several subsystems.



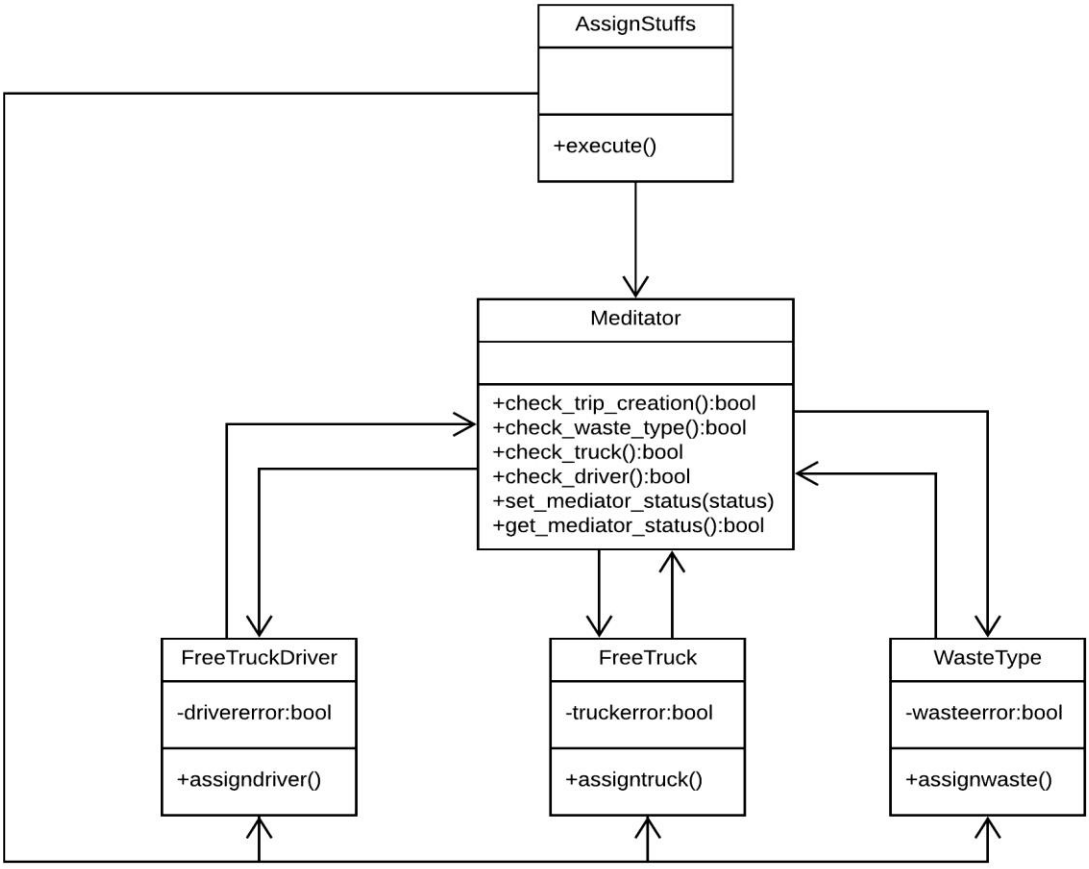
10.6 Mediator Pattern

Definition

The Mediator pattern simplifies communication among objects in a system by introducing a single object that manages message distribution among other objects. The Mediator pattern promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

Implementation

We have used a Mediator class that acts as the common point of contact between FreeTruckDriver, FreeTruck and WasteType classes. The AssignStuffs class instantiates the mediator class and the other three classes. Before each of the three classes execute their methods, they check with the mediator on the status of the other executions before proceeding and after completion they let the mediator know that their execution was completed successfully or unsuccessfully.



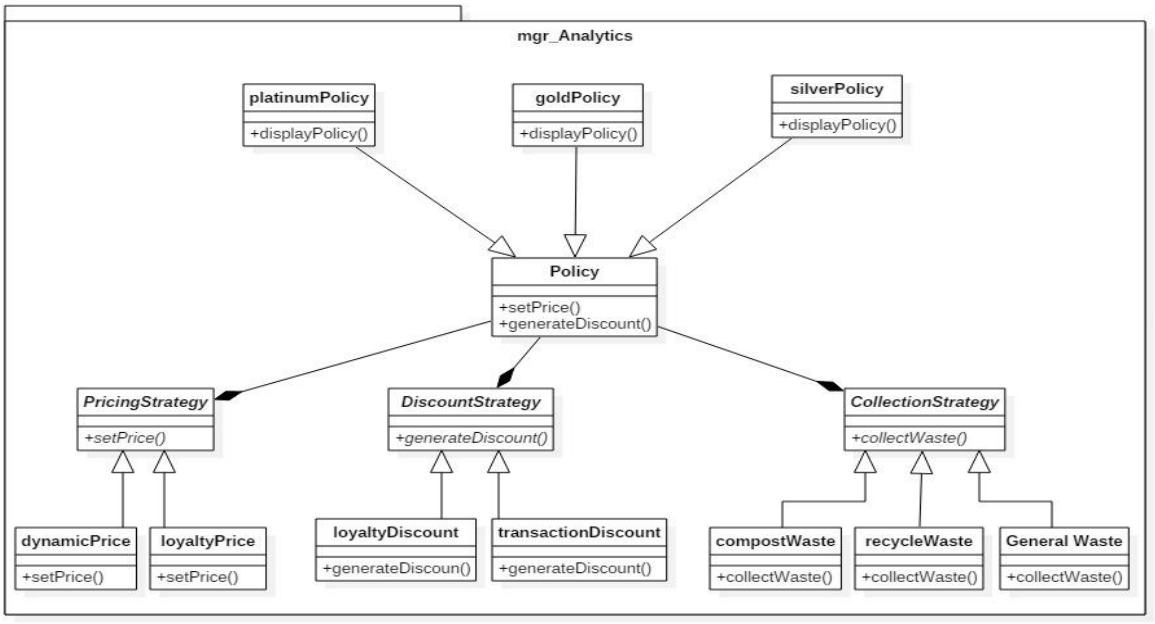
10.7 Strategy Pattern

Definition

The intent of the Strategy Pattern is to define a family of algorithms, encapsulate each algorithm, and make them interchangeable. The Strategy Pattern lets the algorithm vary independently from clients that use it. In addition, the pattern, defines a group of classes that represent a set of possible behaviors. These behaviors can then be used in an application to change its functionality.

Implementation

Scalability and extensibility are two key factors for any business. As part of the bin management system to support these two attributes, we have designed a 'StrategyHub', that houses an array of algorithms which decide the pricing and discounting strategy to be used for different tiers of customers, based on parameters such as transaction value, loyalty factor, and many more. These algorithms are designed as individual classes such as 'platinumPolicy' and 'goldPolicy', which are in turn the concrete implementations of abstract classes such as the 'pricingStrategy' and 'discountStrategy'. The idea is that these algorithms are made available to the clients via an interface class, in our case the 'Policy' class, and at any given point in time the implementation of the concrete classes aka the algorithms' implementation can vary, but the client does not get affected by this, as it is just utilizing the abstraction of these classes. This is a pattern that supports the thought 'program to interfaces, and not implementation'.



11. Added Value

11.1 Django web framework

Our motivation to use Django web framework to build our application was the seamless reflection of MVC architecture. It consists of an object-relational mapper (ORM) that mediates between data models (defined as Python classes) and a relational database ("Model"), a system for processing HTTP requests with a web templating system ("View"), and a regular-expression-based URL dispatcher ("Controller"). We were able to utilize some very interesting features of the framework allowing us to build our application faster than we had originally expected. This includes, but not limited to:

- Class-based views, allowing to structure views and reuse code
- A form serialization and validation system that can translate between HTML forms and values suitable for storage in the database
- An internal dispatcher system that allows components of an application to communicate events to each other via pre-defined signals
- An interface to Python's built-in unit test framework
- Built-in mitigation for cross-site request forgery, cross-site scripting, SQL injection, password cracking and other typical web attacks, most of them turned on by default
- Tools for generating Google Sitemaps

11.2 Object Relational Mapping

By using Django web framework to build our web application, we have harnessed the power of Object Relational Mapping, since Django includes a default object-relational mapping layer (ORM) that can be used to interact with application data from various relational databases. We used MySQL as relational database, but the direct relation between Django models and the corresponding database tables allowed us to make changes to table structure by performing migrations, without the use of queries. Django ORM allowed the team to work fast and produce desired results in a shorter time.

11.3 The Scheduling MechaniX

The heart of the application is the scheduler. A combination of Command and Mediator design patterns is used for its implementation.

Stage 1

All pickup requests from customers are present in the database.

Stage 2

The TripManager class starts execution by adding 3 tasks for execution one after the other namely ScoreRequests (scorer module), ScheduleTrip (scheduler module) and AssignStuffs (assigner module). The sequence starts with ScoreRequests.

Stage 3 - ScoreRequests

It starts with fetching all the pickup requests for the day. Individual scores are calculated based on the weight of the bin to be collected, age of the request and distance of the pickup location from the bin truck home base. The cumulative score for each request is calculated as below:

Cumulative Score = (Age Score + Weight Score – Distance Score) / 3

where

Age Score = Age of the Request / 5

Highest tolerance for pending request is 5 days

Weight Score = Weight of the Bin to be Collected / 50

Highest tolerance for weight is 50 Kg for a Bin type

Distance Score = Distance of the Bin from Home / 10

Highest tolerance for distance is 10 KM for a Trip

Since, we want to minimize the fuel costs, we factor in the distance score negatively in the Cumulative Score. The highest possible cumulative score is assigned to all priority requests for the day. Once all the scores are calculated for a request it is assigned to the request and stored in the database.

Stage 4: ScheduleTrip

This stage starts with setting the weight limit for each truck in the fleet. It starts with fetching all the pickup requests for the day from the database ordered by score in descending and keeps assigning them to the trip until the weight limit of the truck is reached. This continues until all trucks have been filled. All the remaining requests for that day are marked to be picked up for the next day. Notification is sent to all the selected customers.

Stage 5: AssignStuffs

AssignStuffs checks if trip is created and assigns waste type to the trip. Truck is assigned to the trip if trip is created. Driver is assigned to the trip if truck is assigned. It sets the assignment status as True.

Stage 6

The driver is notified of the trip and the assigned truck for the trip.

12. Critique

The project has been a great learning curve for each member in the team. It has given us the opportunity to understand team dynamics, work smartly and understand Software Design in a whole new perspective.
The time spent during the initial phase of the project in understanding each other's strengths played a vital role. It helped us greatly in task allocation allowing team members to play to their respective strengths and collaborate effectively. All programmers were new to using Python for web development and we took it as a challenge in learning Django from scratch.

We found it hard implementing design patterns while using Django and confused it with certain patterns that were built into the syntax of python. Over time, we read, brainstormed and slowly implemented 7 design patterns into various areas of business logic in the system. We learnt to draw several types of UML diagrams and understood how each diagram had clearly outlined and described the system in its own ways. The diagrams recovered from implementation clearly depicted the gaps in original understanding that we had perceived with the system.

While working in Django, we utilized Class-based views wherever possible, that allowed us to structure the code in a better way, allowing systematic reuse of functionalities that are shared between subsystems. Overall, this project opened up new pathways in learning, and we are confident as future software engineers that we have taken a small, but decisive step in putting the things learnt in theory to practice.

13. Appendices

ABBV

Dharaneesh Venkatesh 19018304	DV
Jansirani Subburam 19166125	JS
Kailash Muralidharan 19116608	KM
Kaustab Basu 19005946	KB

Table 1

Package	Class/File	Authors	Lines of Code
mgr_analytics	urls.py, views.py	DV	300+
mgr_accounts	urls.py, view.py, customerFactory.py	KB, KM	300+
mgr_admintask	urls.py, views.py, scheduler.py, scorer.py, manager.py, assigner.py	KB	700+
mgr_database	urls.py, views.py	KM	300+
mgr_notification	urls.py, views.py	JS	200+
mgr_payments	urls.py, views.py	JS	500+
mgr_truckdriver	urls.py, views.py	KM	500+
mgr_customer	urls.py, views.py	DV	400+

Table 2

Student Name	Lines of Code
Dharaneesh Venkatesh 19018304	700+
Jansirani Subburam 19166125	700+
Kailash Muralidharan 19116608	900+
Kaustab Basu 19005946	900+

Table 3

Total Number of Packages	8
Total number of classes and files including GUI implementation	Classes - 65, Files – 40
Total Lines of Code	3200+

13.1 Code Fragments

Command Design Pattern:

```
class TripManager:
        def __init__(self):
        self._commands = []
        def store_command(self,command):
        self._commands.append(command)
        def execute_command(self):
        for command in self._commands:
        command.execute()

class Receiver:
        def action(self):
        print('Starting stage...')

class Manager:
        def main():
        receiver = Receiver()
        stage1 = scorer.ScoreRequests(receiver)
        stage2 = scheduler.ScheduleTrip(receiver)
        stage3 = assigner.AssignStuffs(receiver)
        invoker = TripManager()
        invoker.store_command(stage1)
        invoker.store_command(stage2)
        invoker.store_command(stage3)
        invoker.execute_command()

class ScoreRequests(commander.Command):
        def execute(self):
        self._receiver.action()
        print('This is the scorer !')
        request_list = []
        request_list.append(request1)
        request_list.append(request2)
        request_list.append(request3)
        for requests in request_list:
        scorecalculator = CalculateScore(requests)
        scorecalculator.calculateDistanceScore()
        scorecalculator.calculateAgeScore()
        scorecalculator.calculateWeightScore()
        scorecalculator.calculateTotalScore()
        scorecalculator.displayScore()
```

```
class ScheduleTrip(commander.Command):
        def execute(self):
        self._receiver.action()
        print('This is the scheduler !')
        trip = CreateTrip()
        trip.set_truck_weight_limit()
        trip.fetch_requests_for_today()
        trip.assign_requests_for_route()
        trip.update_requests_status()
        trip.send_notification()

class Command(metaclass=abc.ABCMeta):
        def __init__(self,receiver):
        self._receiver = receiver
        @abc.abstractmethod
        def execute (self):
        pass

class AssignStuffs(commander.Command):
        def execute(self):
        self._receiver.action()
        print('This is the assigner !')
        mediator = Mediator()
        freeDriver  = FreeTruckDriver(mediator)
        freeTruck = FreeTruck(mediator)
        wastetype = WasteType(mediator)
        wastetype.assign_waste()
        freeTruck.assign_truck()
        freeDriver.assign_driver()
        mediator.set_mediator_status()
        print('The Assignment Status is :', mediator.get_mediator_status() )
```

14. References

[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional.

[2] Perdita Stevens, Rob Pooley (1999). *Using Uml: Software Engineering with Objects and Components*, Boston, MA: Addison-Wesley Longman Publishing Co., Inc.

[3] Simon Bennett, Steve McRobb, Ray Farmer (2005). *Object-oriented Systems Analysis and Design Using UML*, McGraw Hill Higher Education.

[4] https://www.djangoproject.com/

[5] https://tutorial.djangogirls.org/

[6] https://sensoneo.com/

[7] https://www.agilealliance.org/

[8] https://www.lucidchart.com/

[9] https://sourcemaking.com/design_patterns

[10]https://python-patterns.guide/