Deliverable #2 Template

SE 3A04: Software Design II – Large System Design

Tutorial Number: T01
Group Number: G6
Group Members:

- Jane Klavir
- Nathan Luong
- Areez Visram
- Jennifer Ye

IMPORTANT NOTES

- Please document any non-standard notations that you may have used
  - *Rule of Thumb*: if you feel there is any doubt surrounding the meaning of your notations, document them
- Some diagrams may be difficult to fit into one page
  - Ensure that the text is readable when printed, or when viewed at 100% on a regular laptop-sized screen.
  - If you need to break a diagram onto multiple pages, please adopt a system of doing so and thoroughly explain how it can be reconnected from one page to the next; if you are unsure about this, please ask about it
- Please submit the latest version of Deliverable 1 with Deliverable 2
  - Indicate any changes you made.
- If you do NOT have a Division of Labour sheet, your deliverable will NOT be marked

1 Introduction

This section should provide an brief overview of the entire document.

1.1 Purpose

State the purpose and intended audience for the document.

1.2 System Description

Give a brief description of the system. This could be a paragraph or two to give some context to this document.

1.3 Overview

Describe what the rest of the document contains and explain how the document is organised (e.g. "In Section 2 we discuss...in Section 3...").

2 Analysis Class Diagram

This section should provide an analysis class diagram for your application.

3 Architectural Design

This section should provide an overview of the overall architectural design of your application. Your overall architecture should show the division of the system into subsystems with high cohesion and low coupling.

3.1 System Architecture

- Identify and explain the overall architecture of your system
- Be sure to clearly state the name of the architecture you used (this is the name of the architectural pattern, not the name of your system)
- Provide the reasoning and justification of the choice of architecture
- Provide a structural architecture diagram showing the relationship among the subsystems (if appropriate)
- List any design alternatives you considered, but eliminated (and explain why you eliminated them)

3.1.1 Overall Architecture and Justification

The overall architecture of the system is the Model-View-Controller (MVC) architecture, which is an interaction oriented architecture. The MVC architecture is the architecture that best fits the system that we are designing. The MVC style is best suited for interactive applications which contain multiple views for various data models. The application being implemented is heavily focused on the UI on the mobile app, and so an architecture style that allows for easy display of data into a UI view is suitable. Furthermore, our system contains multiple data models, for example, models for Users, Rides, Prompts and multiple pages/views across the application will interact with these data models. Furthermore, the MVC architecture is suitable for applications which are prone to frequent data changes. Our system is most definitely prone to frequent data changes, as rides are constantly being offered, arriving, and starting, which means the Rides model will be constantly updated and written to. Finally, the MVC architecture will allow us to separate the various logic functions into separate controllers. The architecture will allow for clear division of logic between controllers, which will make it easier to extend and modify the system. For example, we would have

a UserController to handle user operations, a RidesController to handle ride operations and various other controllers to control the logic of other parts of the application. Given these benefits and the functionality of the system, we have chosen to implement the MVC architecture style for the main system.

3.1.2 Subsystem Architectures and Justification

Another architecture style will also be used for the various subsystems of our application. One of the subsystems within the main system is the Dispatcher subsystem, which is responsible for storing all information on taxis and carpool offerings, as well as deciding how to match carpool offers with requests that come in. The repository architecture style is best suited for systems (in this case a subsystem) in which a central data store stores information and various agents communicate with it. This architecture style fits the Dispatcher subsystem because the Dispatcher will contain a central data store which will store all the information about carpools. The data store will be passive, it will be like a database which just stores information and can be read from. The various agents for the repository will be the users who are requesting and offering carpools. These agents are active, as they drive the flow of the subsystem by sending carpool requests and offering carpools. The agents do not communicate directly with each other, they communicate via the data store by sending requests to the data store, and the data store facilitates the requests and decides appropriate matches. Given this, the repository architecture style is the most suitable for the Dispatcher subsystem.

3.1.3 Structural Architecture Diagram

3.1.4 Design Alternatives Considered

One design alternative for the main architecture that was considered was the Repository architectural style, which is a data centered architecture. We envisioned the system having a central passive database repository, with multiple active agents communicating with the data store. The agents were envisioned to be the different parts of the system, such as a user agent, a rides agent, a prompts agent, and then a dispatcher, all which communicated with the central data store. However, this design was not chosen because the system that is being built is a single mobile application connected to a database. The system does not involve multiple agents which must communicate with a central data store. If there were multiple applications that needed to communicate, this architecture style would be more applicable. However, is is a single application and the different parts of it can be represented using controllers within the architecture rather than external agents, so this design was not chosen.

The second design alternative for the main architecture that was considered was the Presentation-Abstraction-Control (PAC) architecture, which is also an interaction oriented architecture. Both MVC (the architecture that was chosen) and PAC could work for this system, but we ultimately chose MVC over PAC for various reasons. PAC was not chosen because PAC is more useful in complex applications, which require a hierarchical layering of agents. Our system is not complex, it is a single application that communicates with a database. In PAC, the only communication that can occur is between agents, and in our system, we would not have enough agents to justify having communication only between them. This would only add complexity. Furthermore, PAC is more useful for concurrent systems, which our app does not need to be to fulfill its requirements. Finally, PAC is less publicized and less widely used, and so for a somewhat inexperienced development team, MVC is a better choice as there are more resources, information and examples available to help in the design.

3.2 Subsystems

Provide a list of your subsystems, with a brief description of each. Be sure to document its purpose and relationship to other subsystems.

4 Class Responsibility Collaboration (CRC) Cards

This section should contain all of your CRC cards.

- Provide a CRC Card for each identified class
- Please use the format outlined in tutorial, i.e.,

Class Name:	
Responsibility:	**Collaborators:**

A Division of Labour

Include a Division of Labour sheet which indicates the contributions of each team member. This sheet must be signed by all team members.