Team Status 200

RideFind
Software Architecture Document

Version <1.4>

Revision History

Date	Version	Description	Author
19/10/2022	1.0	Initial Creation + Filling in Definitions	William Powers
21/10/2022	1.1	Adding Package Diagram and Definitions	William Powers
23/10/2022	1.2	Add Class Diagram and Fill in Definitions	Joel Clements
23/10/2022	1.3	Final Editing / Formatting	William Powers
30/10/2022	1.4	Format Header, add figures list	Isabel Loney

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using a number of different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions which have been made on the system.

1.2 Scope

This Software Architecture Document provides an architectural overview of the RideFind web application. RideFind allows rideshare users a centralized place to search for rides regardless of ride vendor. It will also allow users to sort and filter drivers based on chosen specifications such as: time estimate, cost of ride, driver rating, car size, etc.

1.3 Definitions, Acronyms, and Abbreviations

Vendor: A company offering ride-share services to the public via their own application interface.

Rideshare: An arrangement in which a passenger travels in a private vehicle driven by its owner, for free or a fee.

Driver: A employee of a rideshare company vendor that uses their car to give others rides and is in turn paid through the vendor.

1.4 References

1. RideFind – Use Case Specifications (Deliverable 4)
2. RideFind – Supplementary Specifications (Deliverable 4)
3. RideFind – Software Requirements Specifications (Deliverable 4)

1.5 Overview

The following in this document contains class diagrams and class definitions for all components of the RideFind web applications.

2. Architectural Representation

This section describes what software architecture is for the current system, and how it is represented. It enumerates the views that are necessary, and for each view, explains what types of model elements it contains.

3. Architectural Goals and Constraints

RideFind will be developed as a web application showcasing three major components: a live map module, and a driver list module, and a page navigation module.

All components must be compatible on any browser engine and utilize Reacts functionality of not reloading when switching pages.

4. Use-Case View

Use-Case View is important to consider when planning a specific iteration centered around a specific use case scenario. It describes the set of scenarios and/or use cases that represent some significant, central functionality as well as describes the set of scenarios and/or use cases that have a substantial architectural

benefit.

Refer to Use_Case_Specification_Document_Status_200.pdf (Deliverable 5) for Use-Case View details.

4.1 Use-Case Realizations

Refer to Use_Case_Realization_Document_Status_200.pdf (Deliverable 5) for all Use-Case Realizations and related models.

5. Logical View

This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities.

5.1 Overview

This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.

5.2 Architecturally Significant Design Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

5.2.1 Design Model: Package Diagrams

The design model represents explicitly the structure and organization of the RideFind system. Following the models, packages are presented with the following specifications: description, corresponding classes, relations, sub packages.

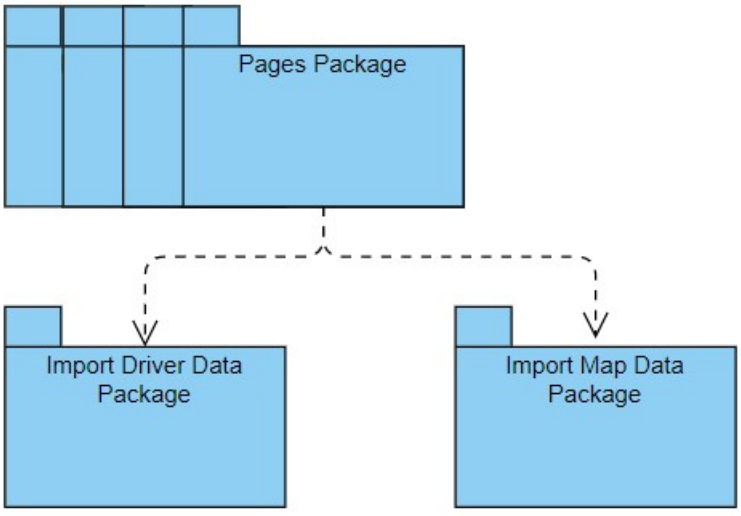


Figure 1: Design Model Packages Level 1

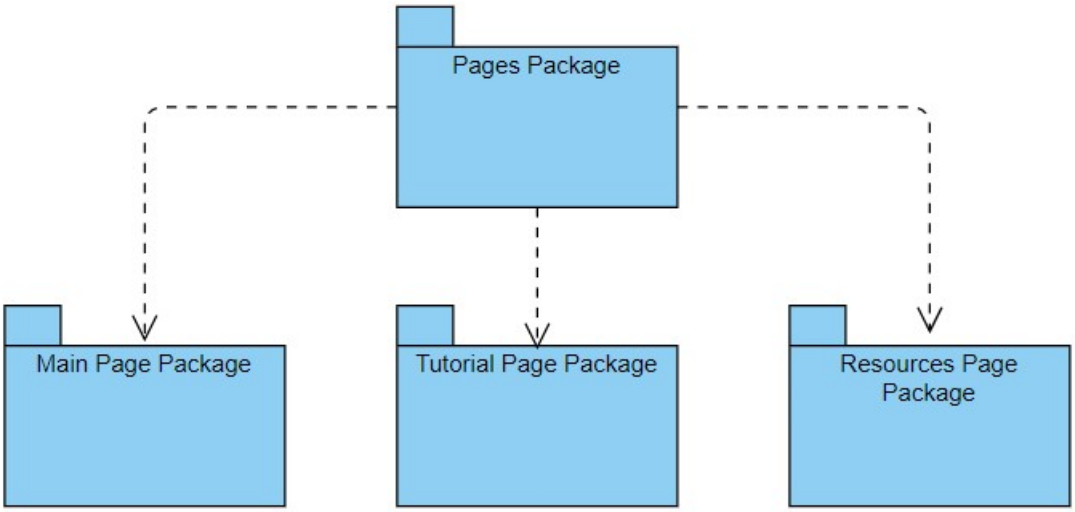


Figure 2: Design Model Packages Level 2

5.2.2 Package Descriptions

Level 1 Packages

Pages	
Description:	The main system package containing all of the separately routed webpages. Excepts input from both the import driver package classes and the import map data package classes to be used and displayed in the sites three pages.
Corresponding Classes:	MainPage, TutorialPage, ResourcePage
Relations:	Main RideFind Package. Dependent on sub packages listed below.
Sub Packages:	Main Page Package, Tutorial Page Package, Resources Page Package.

Import Driver Data	
Description:	Contains all various driver related API classes and deals with all connection authentication to rideshare vendors.
Corresponding Classes:	PullDriverData, DriverAuthentication, CleanDriverData, GetLocalAPIs
Relations:	Dependent on pages package.
Sub Packages:	None.

Import Map Data	
Description:	Contains all various map related API classes and deals with connection authentication to google map vendor.
Corresponding Classes:	MapAuthentication, PullMapData, GetLocalAPIs
Relations:	Dependent on pages package.
Sub Packages:	None.

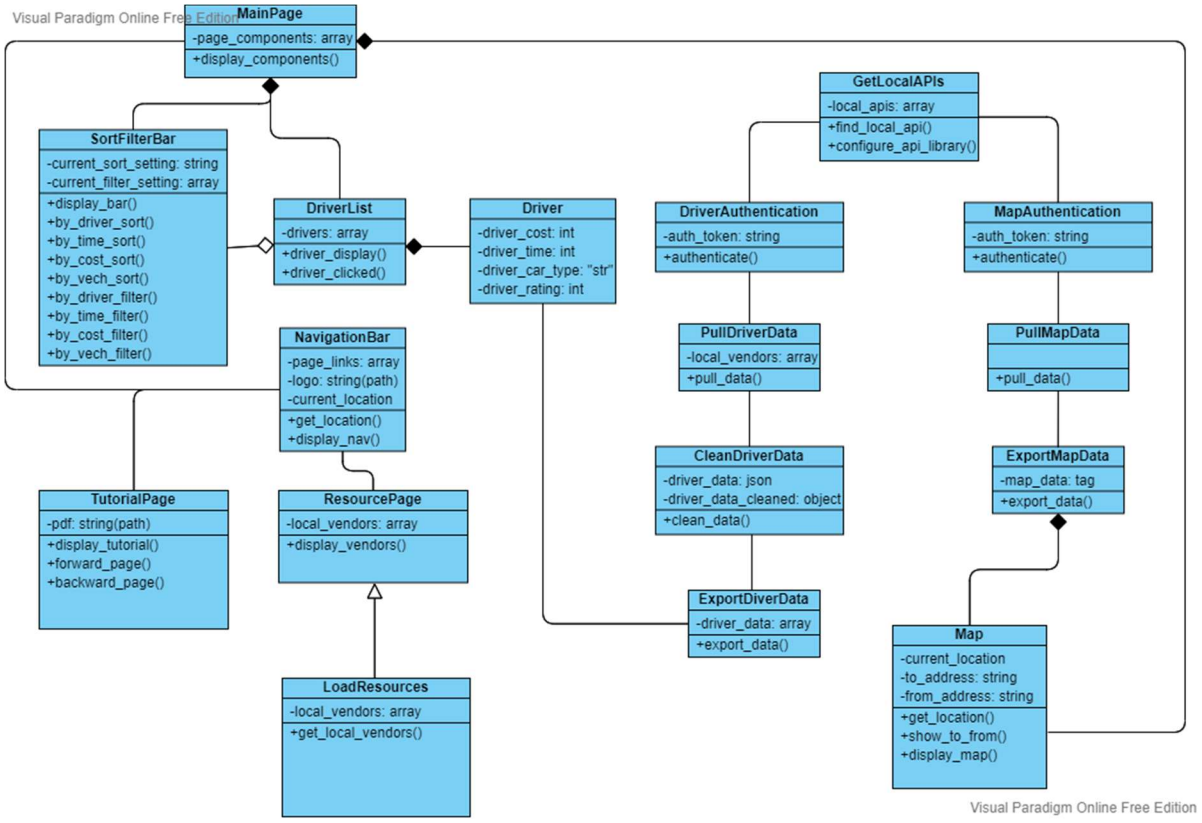
Level 2 Packages

Main Page	
Description:	Contains all classes representing and making up the RideFind home page including the map display, driver list display, and navigation header.
Corresponding Classes:	MainPage
Relations:	Dependent on data received by Pages Package. Shares same navigation module as other Pages Package sub packages.
Sub Packages:	None.

Tutorial Page	
Description:	Contains all classes for viewing and moving through tutorial page.
Corresponding Classes:	TutorialPage
Relations:	Shares same navigation module as other Pages Package sub packages.
Sub Packages:	None.

Resources Page	
Description:	Contains all classes responsible for configuring and displaying location-oriented vendor web application links for user to jump to and from.
Corresponding Classes:	ResourcePage
Relations:	Shares same navigation module as other Pages Package sub packages.
Sub Packages:	None.

5.2.3 Design Model: Class Diagram



5.2.4 Class Description

MainPage	
Description:	Class holding react components of main page display.
Responsibilities:	Display main pages components in correct grid layout.
Relations:	Association to Navigation class, composition to both SortFilterBar class and DriverList class.
Attributes:	-page_components: array
Methods:	+display_components()
Special Requirements:	None.

SortFilterBar	
Description:	Class representing the sort and filter buttons that will change the display of the driver list.
Responsibilities:	Interacting with the driver list class and ordering drivers according to user selected specifications.
Relations:	Aggregation relationship to DriverList class, composition relationship to MainPage class.
Attributes:	-current_sort_setting: string -current_filter_setting: array
Methods:	+display_bar() +by_driver_sort() +by_time_sort() +by_cost_sort(+by_vehicle_sort() +by_driver_filter() +by_time_filter() +by_cost_filter() +by_vehicle_filter()
Special Requirements:	Must be able to view current driver objects in DriverList class.

DriverList	
Description:	Class represents the list of driver classes pulled from vendor APIs.
Responsibilities:	To dynamically load drivers in order selected by user in SortFilterBar class.
Relations:	Aggregation to SortFilerBar class, composition to MainPage class, composition to Driver.
Attributes:	-drivers: array
Methods:	+display_drivers() +driver_clicked()
Special Requirements:	Must adhere to order specifications given by SortFilterBar class.

Driver	
Description:	Class contains each drivers specifications obtained by vendor API pull.
Responsibilities:	Format driver information uniformly for all drivers in order to have clean display in DriverList class.
Relations:	Composition to DriverList class, association to ExportDriverData class.
Attributes:	-driver_cost: int -driver_time: int -driver_car_type: “str” -driver_rating: int
Methods:	None
Special Requirements:	None

TutorialPage	
Description:	Class contains methods to display and navigate tutorial PDF
Responsibilities:	Display a tutorial PDF for new users
Relations:	Association with MainPage, association to NavigationBar
Attributes:	-pdf: string(path)
Methods:	+display_tutorial() +forward_page() +backward_page()
Special Requirements:	None

ResourcesPage	
Description:	Class representing display of extra resources for RideFind users.
Responsibilities:	Show links to local vendor main pages.
Relations:	Generalization to LocalResources classes.
Attributes:	-local_vendors: array
Methods:	+display_vendors()
Special Requirements:	None

LoadResources	
Description:	Class representing the location of and pulling of links for local rideshare vendors.
Responsibilities:	Must be able to find local vendors, and pull their corresponding main webpage links for users to jump to.
Relations:	Generalization to ResourcesPage class.
Attributes:	-local_vendors: array
Methods:	+get_local_vendors()
Special Requirements:	None

GetLocalAPIs	
Description:	Class in charge of locating local rideshare vendors and accessing their prestored API libraries.
Responsibilities:	Must be able to find local vendors and communicate corresponding libraries to DriverAuthenticaiton and MapAuthentication classes.
Relations:	Association to DriverAuthentication and MapAuthentication classes.
Attributes:	-local_apis: array
Methods:	+find_local_apis() +configure_api_library()
Special Requirements:	None

DriverAuthentication	
Description:	Class representing authenticating access to vendors API to obtain local driver information.
Responsibilities:	Must be able to reauthenticate on own when token access expires.
Relations:	Association to GetLocalAPIs class and PullDriverData class.
Attributes:	-auth_token; string
Methods:	+authenticate()
Special Requirements:	Needs to store access token in separate hidden file.

MapAuthentication	
Description:	Class representing authenticating access to Google Maps API to display live map on MainPage class.
Responsibilities:	Must be able to reauthenticate on own when token access expires.
Relations:	Associtaion to GetLocalAPIs class and PullMapData class.
Attributes:	-auth_token: string
Methods:	+authenticate()
Special Requirements:	Needs to store access token in separate hidden file.

PullDriverData	
Description:	Class representing pulling local driver data from vendor API.
Responsibilities:	Must store driver data in clean array of objects to be cleaned and formatted.
Relations:	Association to DriverAuthenticaiton and CleanDriverData classes.
Attributes:	-local_vendors: array
Methods:	+pull_data()
Special Requirements:	None

PullMapData	
Description:	Class representing pulling live Google Map data from API.
Responsibilities:	Must be able to locate current location and center map around that point.
Relations:	Association to MapAuthentication and ExportMapData classes.
Attributes:	
Methods:	+pull_data()
Special Requirements:	None.

CleanDriverData	
Description:	Class in charge of cleaning pulled local driver data into more human-readable objects.
Responsibilities:	Must keep in ordered list according to pulling from API (nearest driver to farthest within local range).
Relations:	Association to PullDriverData and ExportDriverData classes.
Attributes:	-driver_data: array of json objects -driver_data_cleaned: array
Methods:	+clean_data()
Special Requirements:	None.

ExportDriverData	
Description:	Class in charge of cleanly handing local driver data off to the Driver class and then DriverList class.
Responsibilities:	Must keep drivers in order and have clean transfer.
Relations:	Association to CleanDriverData and Driver classes.
Attributes:	-driver_data: array
Methods:	+export_data()
Special Requirements:	None

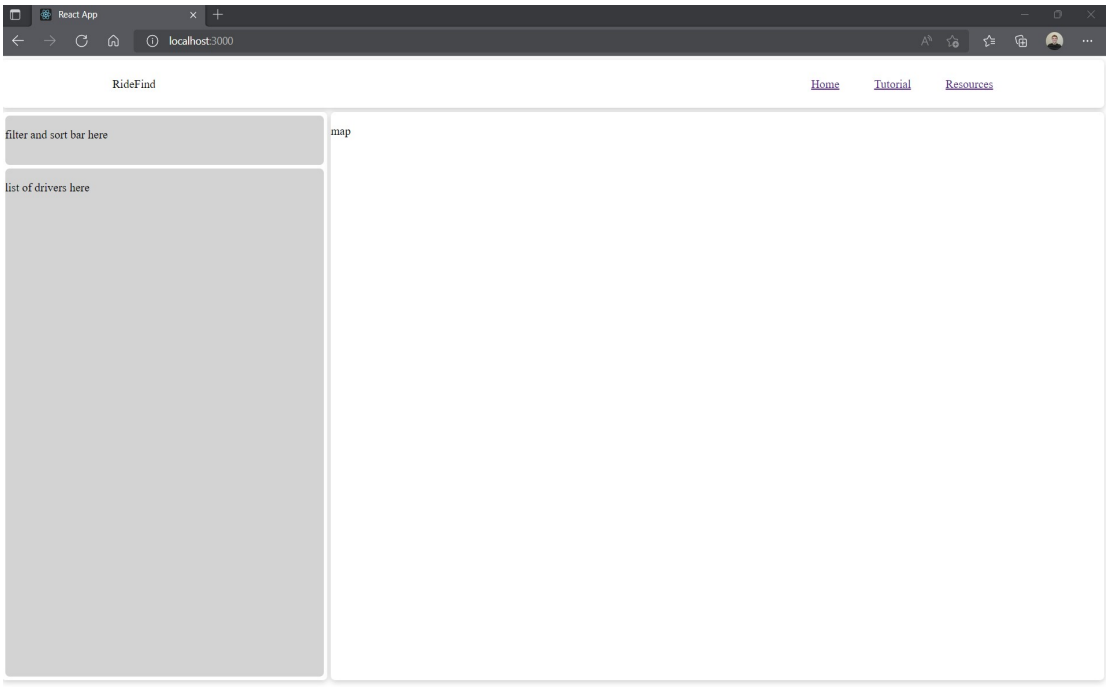
ExportMapData	
Description:	Class in charge of cleanly handing google map data to the Map class withing the MainPage class components.
Responsibilities:	Must keep map centered on current location.
Relations:	Association to PullMapData class, composition to Map class.
Attributes:	-map_data: xml tag
Methods:	+export_data()
Special Requirements:	None.

Map	
Description:	Class representing display on MainPage class of google map centered on current location.
Responsibilities:	Must have all usual map features including zoom, drag, as well as too and from parameters.
Relations:	Composition to ExportMapData and MainPage classes.
Attributes:	-current_location: int -to_address: string -from_address: string
Methods:	+get_location() +show_to_from() +display_map()
Special Requirements:	None.

6. Interface Description

6.1 Overview

The user interfaces focus is ease of use, which being said our goal is to minimize clickable items and bring the few that exist to the focal point. Below is an initial prototype of the user interface in its rawest form.



As seen above the page will consist of three main components; a live map (the right most white outlined div), and ride list (the left most grey outlined div) which includes a sort/filter bar, and the navigation header (the topmost white outlined div). This component layout was modeled from already existing similar map-sort list sites such as Apartments.com and Microsoft Maps in which there is a clean live updating map with a categorical list hovering on one side of the page.

Button ‘Home’ will direct the user to the main page shown, ‘tutorial’ will direct the user to a static page featuring a pdf tutorial put together by the development team, and ‘resources will direct the user to another static page featuring local rideshare vendor webpage links.

The ride list component will feature two pop ups, sort, and filter, which will allow the user to sort and filter the outputted driver list to their liking based off built-in categories including: driver cost, tine estimation, driver rating, vehicle type, etc.

7. Size and Performance

7.1 Overview

The selected framework react will result in slower loading sessions when the user first accesses the site via browser since all pages will be loaded via JavaScript, but very fast when preforming tasks after page is loaded. The only limit will be the speed of outside API connections including the google map API.

There are no size specifications as no information is stored to a server and therefore no database is needed or being used.

8. Quality

8.1 Overview

The software architecture contributes to maximizing ease or use for all users. As outlined above, the main classes and user interface and built to create a high-performance dynamic webpage that intuitively makes sense even for first time users.

API connections to rideshare vendors and google maps will be written and maintained at the highest level as these are the slowest connections on the site and make up the entirety of page loading times. Reliability must also be prioritized for API connections as we must never leave the user with blank ride lists. Even when no drivers are in the area, this must be communicated to the user and subsequently alternative actions must be supplied.

Team Status 200

RideFind
Use-Case-Realization Specification

Version 1.3

Revision History

Date	Version	Description	Author
21/10/2022	1.0	Initial Creation; Introduction section	Isabel Loney
23/10/2022	1.1	Add Sequence Diagrams	Isabel Loney
23/10/2022	1.2	Final Editing / Formatting	William Powers

Table of Contents

Use-Case-Realization Specification

1. Introduction

1.1 Purpose

This document contains information on use-case-realizations for the systems, that being, implementation details for each specified use case. It is intended to capture and convey what objects will be used in the system and how they will collaborate in order to achieve system functionality.

1.2 Definitions, Acronyms, and Abbreviations

Vendor: A company offering ride-share services to the public via their own application interface.

Rideshare: An arrangement in which a passenger travels in a private vehicle driven by its owner, for free or a fee.

Driver: A employee of a rideshare company vendor that uses their car to give others rides and is in turn paid through the vendor.

1.3 References

1. RideFind – Use Case Specifications (Deliverable 4)
2. RideFind – Supplementary Specifications (Deliverable 4)
3. RideFind – Software Requirements Specifications (Deliverable 4)

1.4 Overview

The sections of the Use-Case Realization document describe use cases in terms of their flow of events, participant objects and corresponding diagrams.

2. Use Case 1: First Time User

2.1 Brief Description

Users who are visiting the page for the first time will be given an optional tutorial.

2.2 Flow of Events - Design

Basic Flow: New users are presented with the option to take a tour of the app upon first use, users will be shown how to sort and filter available rideshare options, user will be shown where to access additional resources and help.

Alternative Flow: User may skip tour and access the page as a returning user.

2.3 Interaction Diagrams

Use Case Diagram -> Deliverable 4, Section 2.7
Sequence Diagram -> Deliverable 5, Section 2.3.1

2.3.1 Sequence Diagrams

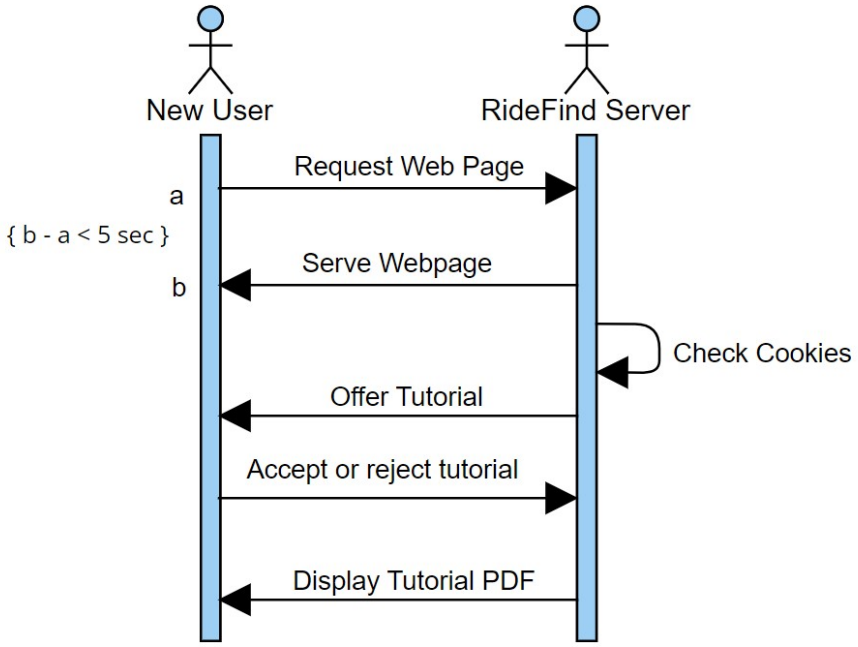


Figure 1: Use Case 1

2.3.2 Participating objects

Object	Description
Tutorial	This represents a pdf with information for new users

2.4 Class Diagrams

Class diagram can be viewed in the Software Architecture Document -> Deliverable 5, Section 5.2.3

3. Use Case 2: Rideshare User

3.1 Brief Description

Returning users that have already completed the tutorial will be able to view local rideshare drivers upon webpage load.

3.2 Flow of Events - Design

Basic Flow: User accesses main page, user inputs their starting and ending location into map, user has option to filter, and sort based on time, cost, vehicle type, and driver rating, when preferred ride is found,

user can select and be sent to organizations site to book.

Alternative Flow: If no rideshare driver available in area user will be notified via driver list.

3.3 Interaction Diagrams

Use Case Diagram -> Deliverable 4, Section 3.5

Sequence Diagram -> Deliverable 5, Section 3.3.1

3.3.1 Sequence Diagrams

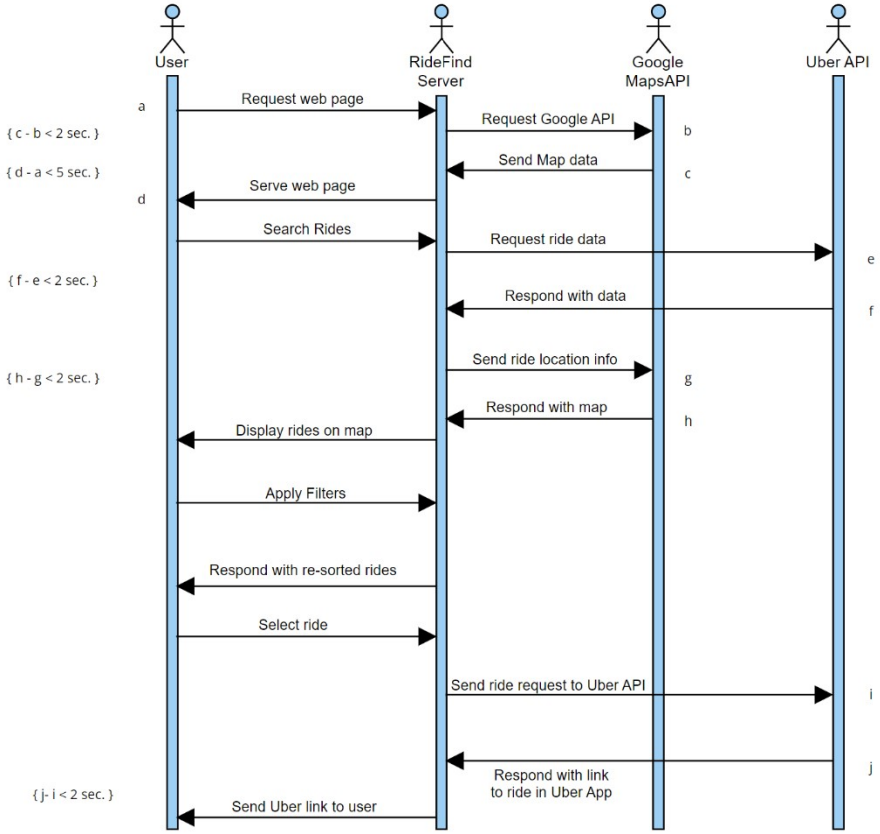


Figure 2: Use Case 2

3.3.2 Participating objects

Object	Description
Main Page	This represents the components of the visible part of the application that allows users to search and view rides
Local APIs	This represents the public APIs of rideshare services that receive requests and respond with data pertaining to a user's desired ride
Google Maps API	This object represents the Google Maps API which provides visual data

	pertaining to a user's location and their desired ride

3.4 Class Diagrams

Class diagram can be viewed in the Software Architecture Document -> Deliverable 5, Section 5.2.3

4. Use Case 3: Commuter <Optional>

4.1 Brief Description

The webpage will include a section dedicated to public transport for non-rideshare users, or user who wish to not use a traditional rideshare driver for their trip.

4.2 Flow of Events - Design

User enters main page, user inputs their starting and ending locations, user can select to view local public transit options that satisfy their trip.

4.3 Interaction Diagrams

Use Case Diagram -> Deliverable 4, Section 4.5

Sequence Diagram -> Deliverable 5, Section 4.3.1

4.3.1 Sequence Diagrams

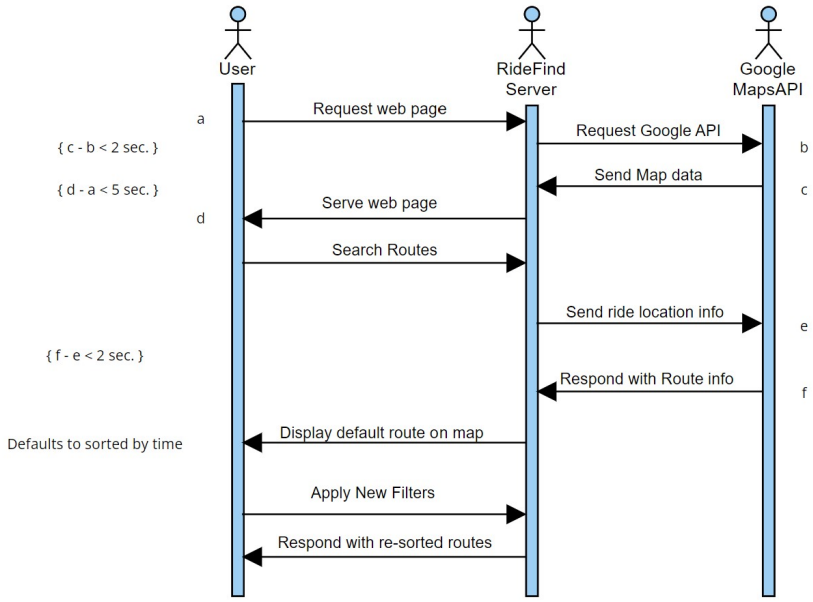


Figure 3: Use Case 3

4.3.2 Participating objects

Object	Description
Main Page	This represents the components of the visible part of the application that allows users to search and view rides
Google Maps API	This object represents the Google Maps API which receives requests and responds with data pertaining to a user's desired public commute

4.4 Class Diagrams

Class diagram can be viewed in the Software Architecture Document -> Deliverable 5, Section 5.2.3