MY TAXI SERVICE

Design Document

Software Engineering 2

Amos Paribocci (854818);Lorenzo Pinosa (852231)

Version 1

Table of contents

1. Introduction

1.1 Purpose

Purpose of this document is to describe the MyTaxiService platform architectural design. Focus on user interface and algorithm design process will also be given.

1.2 Scope

This document describes the design phase which follows the requirements analysis and specification phase in the software lifecycle. MyTaxiService platform scope have already been explained in the RASD.

1.3 Definitions, acronyms and abbreviations

- User: any person who interacts with the system;
- Visitor: any user who has not registered an account or has not logged in;
- Customer/passenger: any user that will exploit the service in order to request a taxi to pick him up;
- Taxi driver: user to which customer requests are forwarded selectively by the system, and is expected to fulfill them in practice;
- Taxi zone: area of the logical city subdivision, to be used for taxi selecting and queueing management while serving a customer request;
- Busy (w.r.t. taxi): taxi that has been assigned a task and is still working on it (picking the customer up or moving towards the ride destination);
- Available (w.r.t. taxi): taxi that is ready to accept a new task;
- App: abbreviation for “mobile application”;
- System/backend: the part of the platform that will hold all the data and do all the processing on it, as well as handle instant communication and notification tasks. Mobile apps and the web service will rely upon this to work;
- Platform: what all MyTaxiService components build up, including the Website, the mobile apps and the backend;
- RASD: Requirements Analysis and Specification Document.

1.4 References

- Template for the design document (Software Engineering 2 course);
- ISO/IEC/IEEE 42010: Systems and software engineering – Architecture description;
- IEEE Standard for Information Technology – Systems Design – Software Design Descriptions;
- What Brand Colors Say About Your Business – Marketo (http://blog.marketo.com/2012/06/true-colors-what-your-brand-colors-say-about-your-business.html)

1.5 Document structure

This document follows the template given in the context of Software Engineering 2 course.

Section 2 will cover the platform architectural design, showing how components interact with each other, the provided interfaces and the used architectural patterns.

In section 3 we will provide an example algorithm w.r.t. one of the product functionalities.

Section 4 will feature information about the user interface design.

Lastly, section 5 will explain how the design directives written here are consistent with the RASD content.

2. Architectural design

2.1 Overview

We aim to provide an extensible and flexible yet robust structure, while keeping an eye on maintainability. That's why we have chosen a modular architecture for MyTaxiService backend, and we are oriented to a Service Oriented Architecture to allow further extensions of the system and provide information to them. System Backend communication with both mobile and web app will be according to a client/server pattern for user requests; a publisher/subscriber pattern is used to dispatch notifications to mobile apps.

2.2 High level components and their interaction

2.2.1 System Backend

The System Backend will deal with all background tasks needed in order to run the entire platform. It will receive and send data to mobile & web apps, manage the database, do all the required processing and so on.

2.2.2 Mobile Apps and Web Service

The final interaction with the user will be carried out by the Mobile Apps and the Web Service. There will be apps for the most used mobile OSes and a responsive Web Service (as stated in the RASD).

2.3 Component view

2.3.1 System Backend

The System Backend will be split in the following modules: Request Handler Module, Communication Module, Taxi Location Handler Module, Taxi Selection Module, Notification Module, Taxi Booking Management Module.

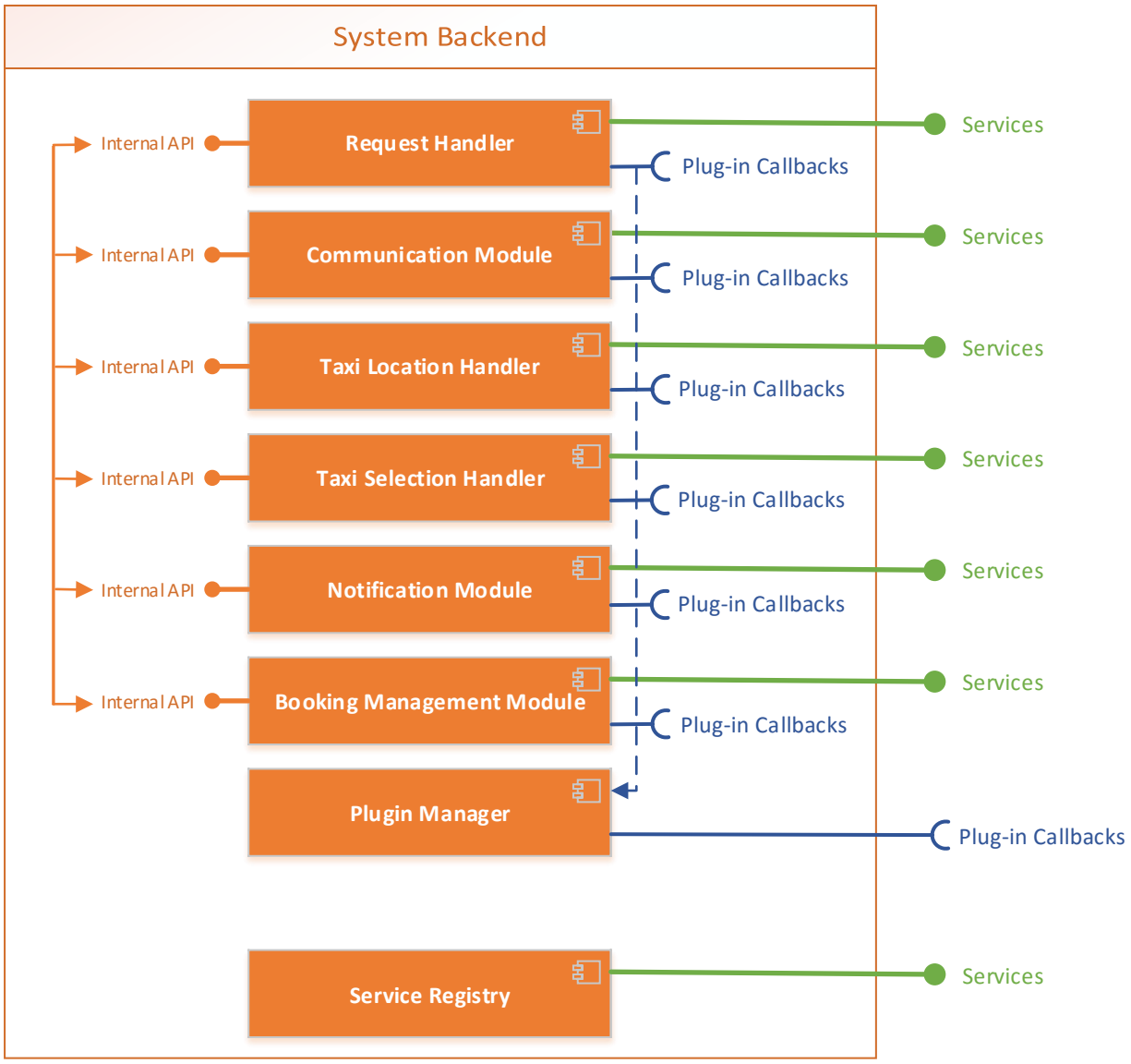
As stated in the RASD document, we want MyTaxiService to be extensible in the future, even by exploiting specifically engineered plugins.

Plug-in pattern will obviously be followed: there will be a Plug-in Manager Module that will load plug-ins and pass them the Plug-in Callbacks that each of the previous specified modules will expose, allowing to hook some or all the functionalities provided by them.

Plug-ins may need some information from other system backend modules than the one that triggered the Plug-in Callback, as well as the system backend may be modified in the future to exploit some data or elaboration provided by plug-ins. To easily meet with these situations, we decided to embrace also a Service Oriented Architecture: each of the modules will expose also a set of services, as well as any plugin can do (but has not to); a Service Registry Module will also be added to the system.

The resulting architecture is illustrated in Figure 2.3.1-1; the dashed arrow connecting the Modules Plug-in Callbacks to the Plug-in Manager Module means that these callbacks are not directly exposed to plug-ins for security and flexibility reasons but are proxied by the Manager.

Figure 1 The system backend modular structure



2.3.1.1 Request Handler Module

This is the main module of the system backend: it exploits and coordinates all other modules in order to fulfill all System Backend tasks. It's activated mainly by the communication module (See 2.3.1.2) which forwards every user request or update to be processed; for example, a location update of a taxi, or a taxi request by a customer (see the Taxi Request Sequence Diagram on the RASD document). Its methods are called for example also by the Booking Management Module, when a taxi booking is due and a taxi request should be issued.

2.3.1.2 Communication Module

This module will handle communication with mobile and web apps following a client/server approach. We searched a standardized, reliable and widespread protocol to exploit and we have chosen to deploy a HTTP RESTful service. Duty of this module is also to offer to the notification one abstracting methods to send notifications via XMPP protocol and provide back errors and results.

2.3.1.3 Taxi Location Handler Module

As far as spatial data elaboration is concerned, the processing is handled by this module. It is in charge of determining taxi and customer locations with respect to taxi zones and of building the list of taxi suitable for addressing a specific customer request.

2.3.1.4 Taxi Selection Module

Here are implemented all the algorithms which take care of picking the best taxi from a list of suitable ones, as well as providing another as fallback in case the first is not available. This module might have a high computational load impact on the database since it needs to build statistics and access many data in order to carry out this tasks.

2.3.1.5 Notification module

This is a small module which provides notification facilities to the system. It deals with reliable message dispatching, handling errors such as delivery ones while taking care of determining whether a recipient is available or not. This information is very useful to the system, for taxi selection algorithm for example to automatically exclude not available taxis, and it is persisted in the database. Searching for a platform which offers notification services we handpicked Google Cloud Messaging, which is free for unlimited messages and relies on XMPP protocol thus providing delivery receipt and even upstream messages (from device to server); as stated in 2.3.1.2, it will be the Communication Module to deal with this protocol providing a level of abstraction to the Notification one.

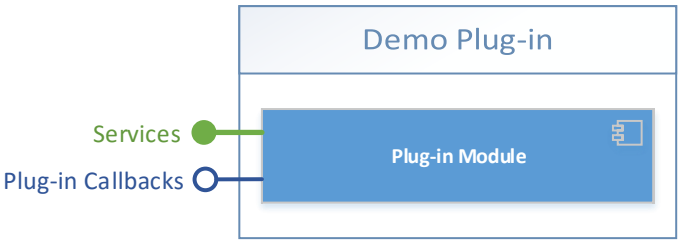
2.3.1.6 Taxi Booking Management Module

Booking management module will look after user booking requests. It will be notified from the Request Handler when a customer issue a new one and it will take care of forwarding it to the Request Handler again at proper moment. This module should deal also with booking modifications and canceling. See the Taxi Booking Request Sequence Diagram on the RASD document for more details about this process.

2.3.1.7 Plug-in Manager Module

This module will deal and interface with MyTaxiService plug-ins. It will take care of identifying compatible plugins when the system starts, load them, and pass them all the plug-in callbacks triggered by the other modules.

A demo plug-in structure can be viewed in the image at the right.



2.3.1.8 Service Registry

A Registry to get the SOA (Service Oriented Architecture) working.

2.3.2 Mobile Apps and Web Service

All user interactive components of MyTaxiService won't carry out any relevant computational task and thus will be mainly split into two modules:

2.3.2.1 Communication module

This will handle downstream and upstream communication with the server. It will interface with the System Backend communication module forwarding it user requests and status updates and will receive data originating from the System Backend notification module.

2.3.2.2 User interface module

All user interaction with MyTaxiService will go through this module: graphical input controls will be handled at this level as well as data and response from the server presentation.

2.4 Deployment view

2.4.1 System Backend

The deployment view of the System Backend is pretty easy: once ready, it will be installed on a server machine. The Operating System running the backend will for sure be virtualized, to gain all the benefits that such technology guarantees.

2.4.2 Mobile Apps

The Mobile Apps will be deployed by uploading them to the App Stores of each mobile OS.

2.4.3 Web Service

The Web Service will be deployed on the same Web Server hosting MyTaxiService website. The Web Server is a virtual machine initially deployed to the same physical hardware hosting the System Backend, it will be moved in future to dedicated HW if the workload is expected to be too high.

2.5 Runtime view

The sequence diagrams describing the taxi booking and taxi reservation procedures have been already presented in the RASD Document.

2.5.1 Plug-in Runtime View

To better clarify how plug-ins will interact with the System Backend, we have produced two sequence diagram showing how a plug-in is initialized by the Plug-in Manager Module at System startup, and how are working the plug-in callbacks proxied by the Plug-in Manager.

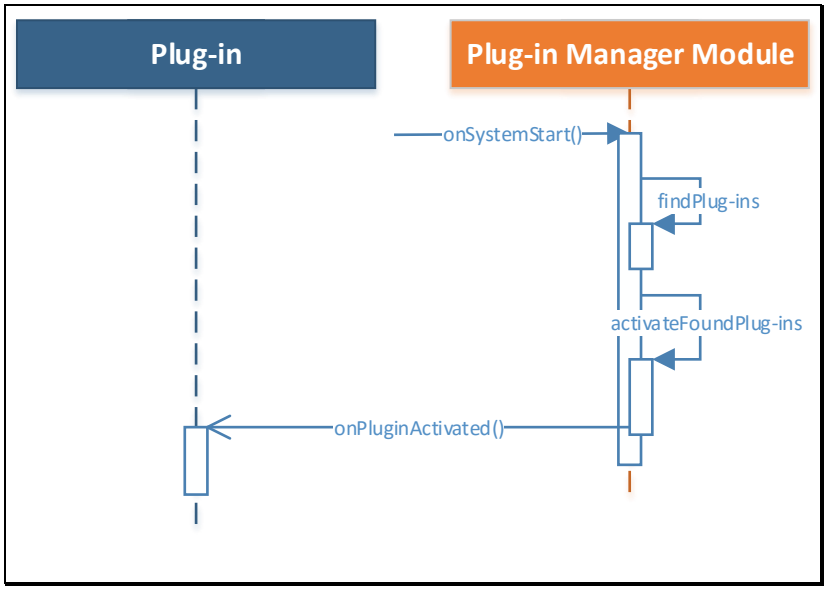


Figure 2 Plug-in Initialization

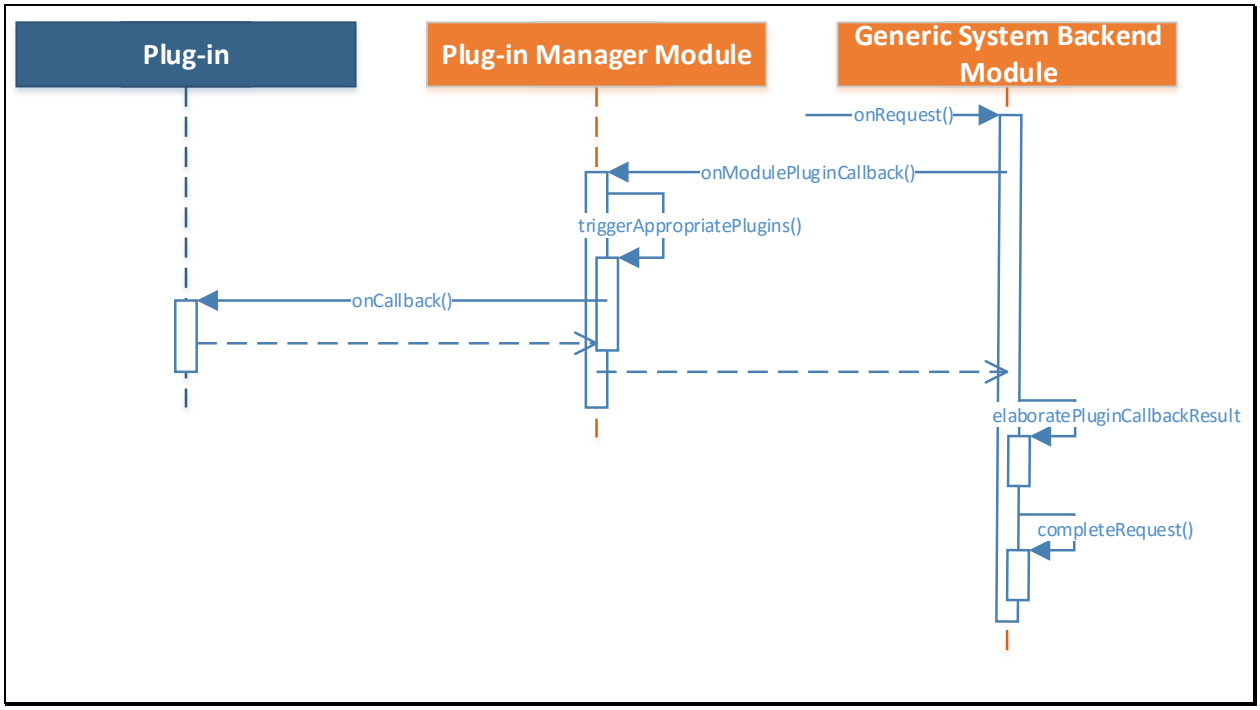


Figure 3 Plug-in Callbacks Proxied by Plug-in Manager Module

2.6 Component interfaces

In this section we will list the main methods that each module will expose as Internal API and Services, as well as the ones it will callback on available plug-ins. As described in section 2.5.1, all callbacks to plug-ins listed here will be incapsulated and forwarded to Plug-in Manager Module via onModulePluginCallback().

2.6.1 Request Handler Module

2.6.1.1 Internal API

- `createNewRequest(Customer, Source, Destination)`
- `createNewBooking(Customer, Source, Destination, DateTime)`
- `removeBookingRequest(Id)`
- `modifyBooking(Id, Source, Destination, DateTime)`
- `getBookingRequestList(Customer) returns List<Booking>`
- `updateTaxiPosition(Taxi, Location)`

2.6.1.2 Plug-in Callbacks

The Boolean returned indicates whether the plug-in has already handled the task and thus overridden the default behavior or not. If the plug-in returns true, the module will do nothing to handle the task assigned (for which this plug-in callback has been made), considering it already fulfilled.

- `onNewRequest(Customer, Source, Destination) return Bool`
- `onNewBooking(Customer, Source, Destination, DateTime) return Bool`

2.6.1.3 Services

- `createNewRequest(Customer, Source, Destination)`
- `createNewBooking(Customer, Source, Destination, DateTime)`

2.6.2 Communication Module

2.6.2.1 Internal API

- `sendNotification(Recipient, MessageObject) returns Result`

2.6.2.2 Plug-in Callbacks

The Boolean returned indicates whether the plug-in has already handled the task and thus overridden the default behavior or not. If the plug-in returns true, the module will do nothing to handle the task assigned (for which this plug-in callback has been made), considering it already fulfilled.

- `onMessageFromCustomerRecieved(Customer, MessageObject) return Bool`
- `onMessageFromTaxiDriverRecieved(TaxiDriver, MessageObject) return Bool`
- `onSendingNotification(Recipient, MessageObject) return Bool`

The MessageObject class incapsulates a message, including information such as the type (i.e. a taxi request, a taxi booking…), an eventual user message, a timestamp and so on.

2.6.2.3 Services

No external services will be provided in the initial release.

2.6.3 Taxi Location Handler Module

2.6.3.1 Internal API

- `findTaxis(TaxiZone) returns List<Taxi>`
- `getZoneFromLocation(Location) returns TaxiZone`
- `updateTaxiPosition(Taxi, Location)`

2.6.3.2 Plug-in Callbacks

The Boolean returned indicates whether the plug-in has already handled the task and thus overridden the default behavior or not. If the plug-in returns true, the module will do nothing to handle the task assigned (for which this plug-in callback has been made), considering it already fulfilled.

- `onFindTaxis(TaxiZone) returns List<Taxi>`
- `onGetZoneFromLocation(Location) returns TaxiZone`
- `onRetrieveTaxiZones() returns List<TaxiZone>`
- `onUpdateTaxiPosition(Taxi, Location) returns Bool`

2.6.3.3 Services

- `findTaxis(TaxiZone) returns List<Taxi>`
- `getZoneFromLocation(Location) returns TaxiZone`

2.6.4 Taxi Selection Module

2.6.4.1 Internal API

- `selectTaxi(TaxiZone, TaxiRequest) returns Taxi`

2.6.4.2 Plug-in Callbacks

- `onSelectTaxi(TaxiZone, TaxiRequest) returns Taxi`
- `onGetSelectionAlgorithm() returns ITaxiSelectionAlgorithm`

2.6.4.3 Services

- `selectTaxi(TaxiZone, TaxiRequest) returns Taxi`

2.6.5 Notification module

2.6.5.1 Internal API

- `sendAskConfirmationNotification(TaxiDriver, TaxiRequest)`
- `sendTaxiIncomingNotification(Customer, TaxiRequest)`
- `sendNoTaxiNotification(Customer, TaxiRequest)`

2.6.5.2 Plug-in Callbacks

The Boolean returned indicates whether the plug-in has already handled the task and thus overridden the default behavior or not. If the plug-in returns true, the module will do nothing to handle the task assigned (for which this plug-in callback has been made), considering it already fulfilled.

- `onSendAskConfirmationNotification(TaxiDriver, TaxiRequest) return Bool`
- `onSendTaxiIncomingNotification(Customer, TaxiRequest) return Bool`
- `onSendNoTaxiNotification(Customer, TaxiRequest) return Bool`

2.6.5.3 Services

- `sendNotificationToCustomer(Customer, Text)`

2.6.6 Taxi Booking Management Module

2.6.6.1 Internal API

- `addBookingRequest(Customer, Source, Destination, DateTime)`
- `removeBookingRequest(Id)`
- `modifyBooking(Id, Source, Destination, DateTime)`
- `getBookingRequestList(Customer) returns List<Booking>`

2.6.6.2 Plug-in Callbacks

The Boolean returned indicates whether the plug-in has already handled the task and thus overridden the default behavior or not. If the plug-in returns true, the module will do nothing to handle the task assigned (for which this plug-in callback has been made), considering it already fulfilled.

- `onAddBookingRequest(Customer, Source, Destination, DateTime) return Bool`
- `onRemoveBookingRequest(Id) return Bool`
- `onModifyBooking(Id, Source, Destination, DateTime) return Bool`
- `onGetBookingRequestList(Customer) returns List<Booking>`

2.6.6.3 Services

- `removeBookingRequest(Id)`
- `modifyBooking(Id, Source, Destination, DateTime)`
- `getBookingRequestList(Customer) returns List<Booking>`

2.6.7 Plug-in Manager Module

2.6.7.1 Internal API

- `onModulePluginCallback(PluginCallback)`

2.6.7.2 Plug-in Callbacks

We don't want the Plug-in Manager Module to be extended. The "Plug-in Callbacks" extending outside of the System Backend boundary in Figure 1 means that this Module will proxy all Plug-in Callbacks.

2.6.7.3 Services

- `getAvailablePluginList() returns List<Plugin>`

- `getActivePluginList() returns List<Plugin>`

2.6.8 Service Registry

2.6.8.1 Internal API

- `searchService(ServiceType type)`

2.6.8.2 Plug-in Callbacks

The service registry will not be extensible.

2.6.8.3 Services

- `searchService(ServiceType type)`

2.7 Selected architectural styles and patterns

Designing and architecting MyTaxiService we followed some popular architectural pattern; they are described in section 2.6.1 and we recap them here:

- SOA – Service Oriented Architecture
- Plug-in architecture to guarantee extendibility
- Client/server
- Publisher/subscriber communication protocol.

3. Algorithm design

In this chapter we will expose some example algorithm featured in the System Backend.

3.1 Request Handler

The following code runs when the RequestHandler component receives a new taxi request: using the internal API provided by TaxiLocationHandler and TaxiSelectionHandler components, a taxi is selected for the task and a new request is created. If no available taxi is found, an appropriate notification is sent to the customer, using the NotificationModule.

```
void createNewRequest( Customer c, Location src, Location dst ) {
	// assumption: input parameters are correct and
	// consistent with the software database
	Integer id = getNextTaxiRequestId();
	TimeStamp time = System.getCurrentTimeStamp();
	TaxiRide ride = null; //will be created after the ride is finished
	TaxiRequest request = new TaxiRequest( id, src, dst, c, time, ride );
	storeTaxiRequest( request );
	TaxiZone zone = TaxiLocationHandler.getZoneFromLocation( src );
	List<Taxi> taxiList = TaxiLocationHandler.findTaxis(zone );
	Taxi taxi = TaxiSelectionHandler.selectTaxi(zone, request);
	if ( taxi == null ) {
		// no available taxi has been found
		NotificationModule.sendNoTaxiNotification( c );
		return;
	}
	NotificationModule.sendTaxiIncomingNotification(c, request)

}
```

3.2 Taxi Location Handler

The code listed below is used by the TaxiLocationHandler for getting the list of available taxis, given the TaxiZone. An auxiliary method allows the system to get the containing TaxiZone, given the Location coordinates.

```
List<Taxi> findTaxis(TaxiZone zone) {
	List<Taxi> taxiList = new ArrayList<Taxi>();
	for (Taxi taxi : taxiDatabase) {
		if (taxi.isAvailable()) {
			TaxiZone z = TaxiLocationHandler.getZoneFromLocation(
			taxi.getPosition());
			if (z == taxi.taxiZone)
				taxiList.add(taxi);
```

```
            }
        }
        return taxiList;
}

TaxiZone getZoneFromLocation(Location l) {
        for (TaxiZone zone : zoneDatabase)
            if (zone.geoShape.contains(l)) //exploit database geography
capabilities (through an ORM here)
                    return zone;
        return null;
}
```

4. User interface design

4.1 UI Design process

We followed the “know your audience” advice and started by having in mind our users as depicted in the RASD scenarios; the result are the following personas.

4.1.1 Personas

4.1.1.1 Giulio

Giulio is 22 years old and is studying at the university in a big city. He is away from home and he relies on his parents for economic support. He travels by train, sometimes by plane, and as many other students he doesn’t own a car. Giulio is passionate about technology: he loves following the last trends and owning hi-tech devices.

Since Giulio is always trying new applications, he prefers to have an immediate overview (or a quick tour) of the offered features without actually having to register new accounts or to fill out forms.

4.1.1.2 Rosy

Rosy has recently graduated and is now working as risk analyst for a financial company. She is 26 years old and economically independent, but mainly relies on public transportation or taxi service in order to accomplish her duties. Rosy likes to show a professional attitude and tends to choose services that look reliable and trustworthy.

Rosy is a very active, dynamic and fast-paced person, always striving for productivity: she does not feel like devoting much time to technology usage.

4.1.1.3 Luca

Luca is 31 years old and works as an employee at the local supermarket: he owns a car but uses the taxi service from time to time. Since Luca tends to be very clumsy and distracted, he has got bad experiences with taxis: a few years ago he even forgot his wallet in the taxi and had to go through a long process to have it back. He is a relaxed, easygoing person and has got many friends overseas.

4.1.1.4 Steve

Steve is a new taxi driver at a big taxi agency in the city: he is 27 and has been recently hired.

He is a concrete and straightforward person, now trying to build a family with his partner. Steve is definitely not a technology guy, since he uses his laptop and smartphone for performing simple tasks only: his favorite products are robust and easy to use.

4.1.2 Design decisions and consequences

In this paragraph we aim to explain some design decisions regarding the MyTaxiService UI, regarding both the mobile app and the Web application.

The user interface will feature the following characteristics.

4.1.2.1 Web & Mobile app – Making the taxi map available to the visitors

When the application (in both Web and mobile versions) is opened by a visitor, the city map with appropriate icons for taxi positions should be displayed. This will provide the users an immediate idea about the way the service is provided and that it is really up and working, even before registering a new account. Giulio would really like this and get involved, if he had to register an account before seeing anything he could have lost interest in our service.

4.1.2.2 Web & mobile app – Choosing the right color palette

In order to establish the color palette for our products, we relied on basic color theory.

It is well known that different colors stimulate different sensations and feelings: for example, red is associated with power and energy, while blue inspires trust and security. Such stimulation may lead to better communication and product branding.

Among all the colors, both red and yellow suggest ideas like motivation, dynamism, movement, action and speed: we believe that this perfectly match our service characteristics.

In order to create some contrast, encourage call-to-action (i.e. having the users clicking on a particular button) and put emphasis on specific elements, the triad color rule will be used.

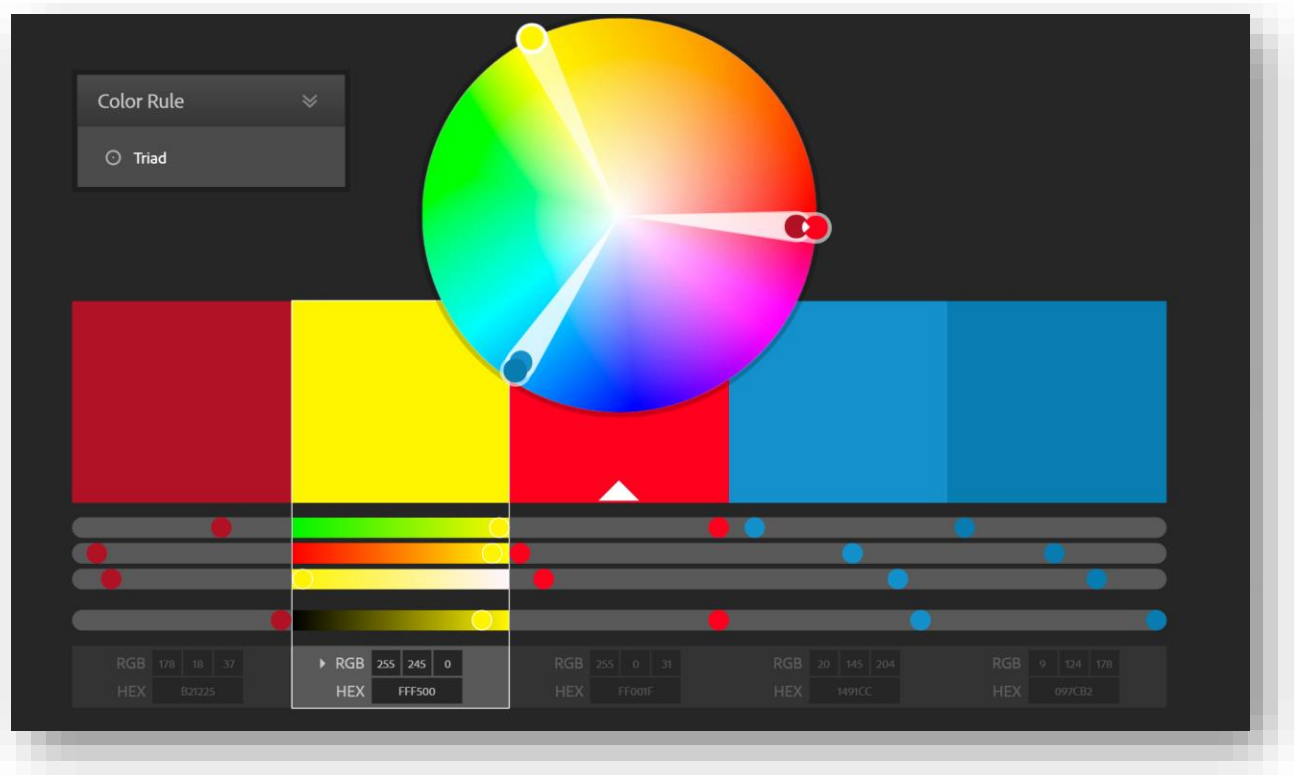


Figure 4.1.2.2 Color palette specification using triad color rule

The triad color rule is used to individuate two colors (given a basic color, yellow in our case) to be used to create contrast and interest. When highlighted on the color wheel, the three colors form a triangle (as shown in figure). The triad color rule is considered a "safe rule": it allows designers to give a vibrant yet acceptable look to Web pages and app screens, without taking too much risk (like when using the complementary color rule, risking to stress the user eyes too much by using too much contrast).

We think this applies fairly well to our requirements.

4.1.2.3 Web app – Catching eye attention with the Z-Pattern

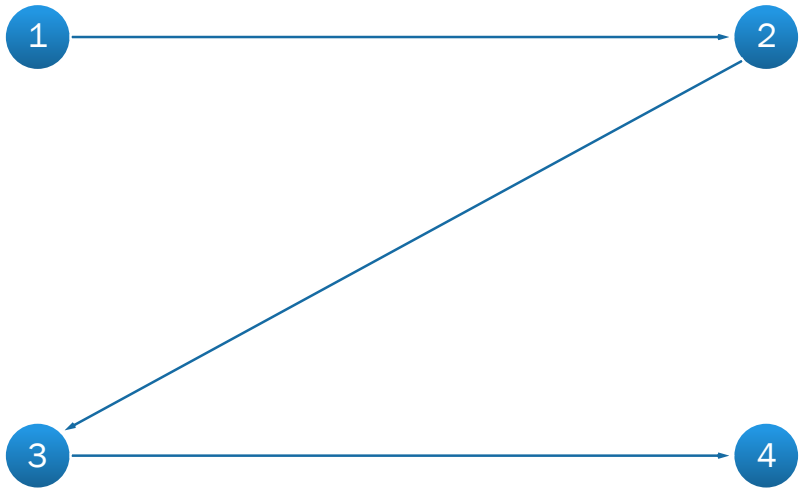


Figure 5.1.2.3 Showing the typical Z-Pattern flow

Color is not the only visual concept that influences the users' behavior when navigating through Web pages or app screens: element disposition and visual layouts are crucial for a good user experience, as well.

In order to deliver an effective communication and to put emphasis on the right page elements, the visual Z-Pattern (or Z-Layout) will be used. This typical web design layout, together with other visual patterns like the F-Pattern, is used to impose a shape (a "Z" shape, in this case) on the page: by positioning the page elements along the shape path, the user eyes are expected to flow along such lines (starting at point 1, then going to points 2, 3 and 4, as shown in the figure).

This visual pattern is also used to stimulate call-to-action: a typical application consists in putting a contextual menu along line 1→2, ending with an action element (like a log-in form or sign up button) on point 2.

4.1.2.4 Mobile app – Offering new users a quick tour

The MyTaxiService mobile application will feature a "quick tour" when opened for the first time: a scroll panel view will display basic information about the most important aspects of the app, thus explaining the way to use it. Steve will love this: finally an app that explains itself and no more time wasted trying to figure out everything.

4.1.2.5 Mobile app – Ensuring immediate assistance

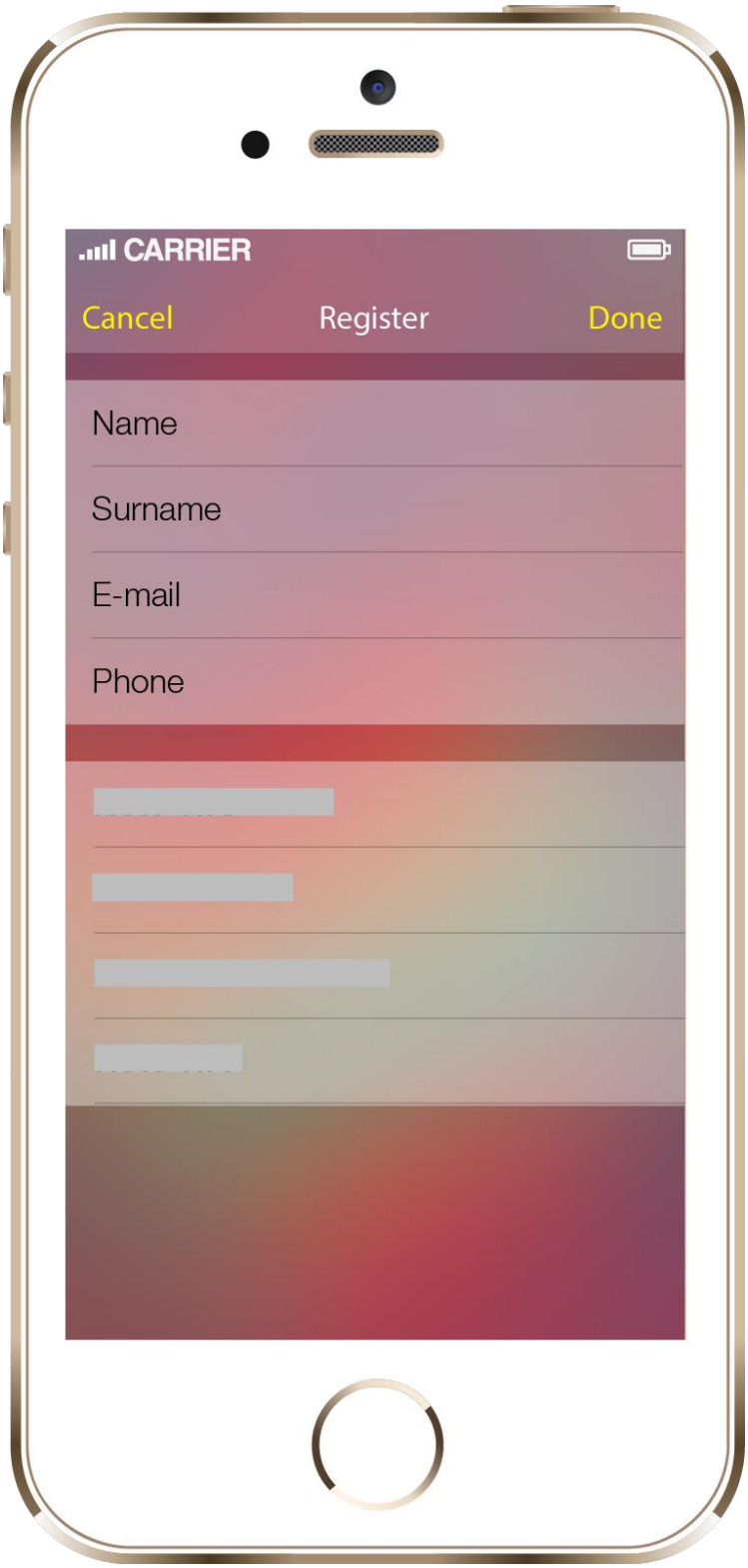
We want to allow our mobile users to receive support as promptly as possible. That is why the MyTaxiService mobile app will offer a simple and intuitive assistance screen, letting our users decide whether calling for support or sending an email. An appropriate assistance infrastructure shall be present on the company side. Luca will no more have to spend days to get his lost wallet back: as soon

as he realizes what happened, with two taps on MyTaxiService app he will get in touch with the customer service which will contact the taxi driver – exploiting the system records.

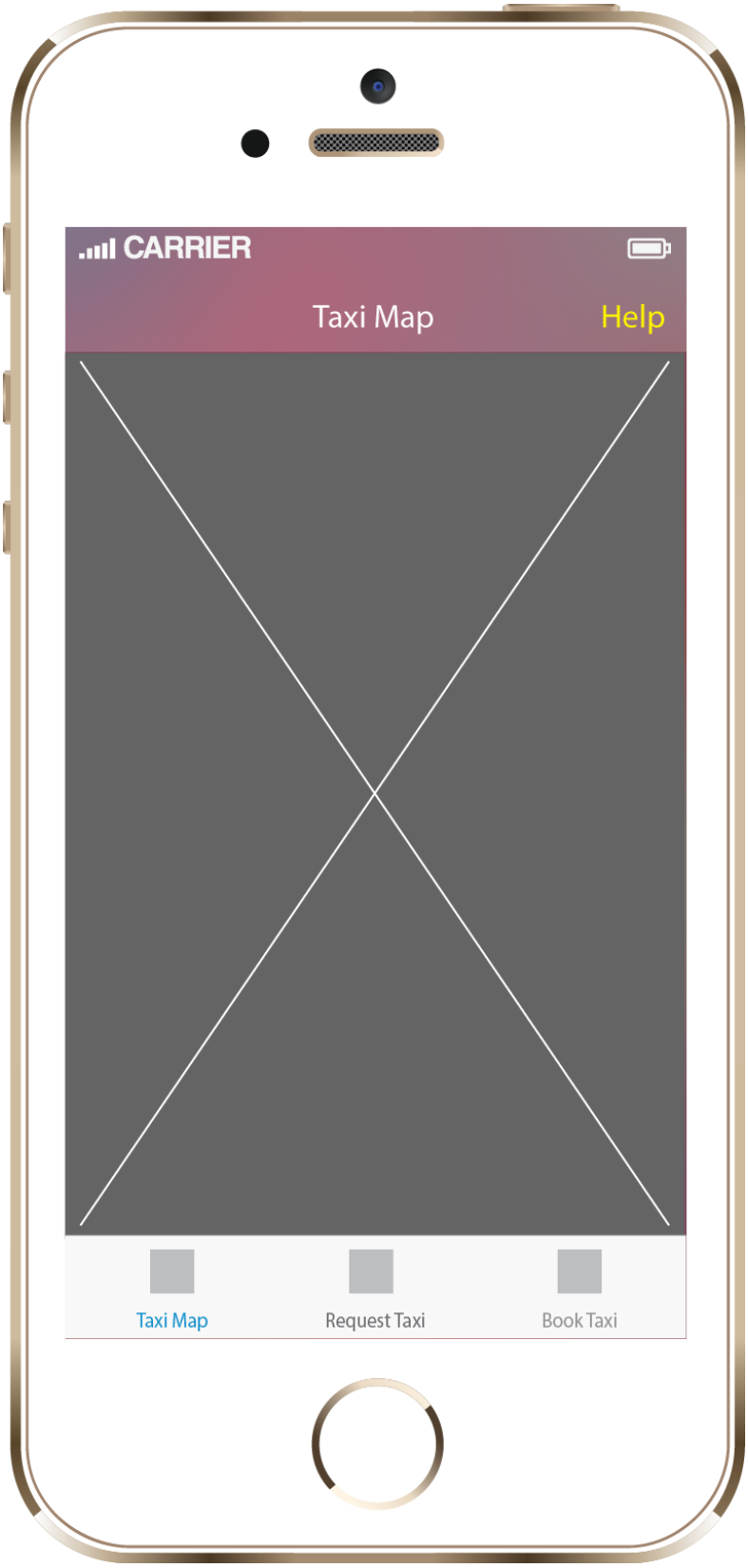
4.2 UI mockups

4.2.1 Customer Mobile App

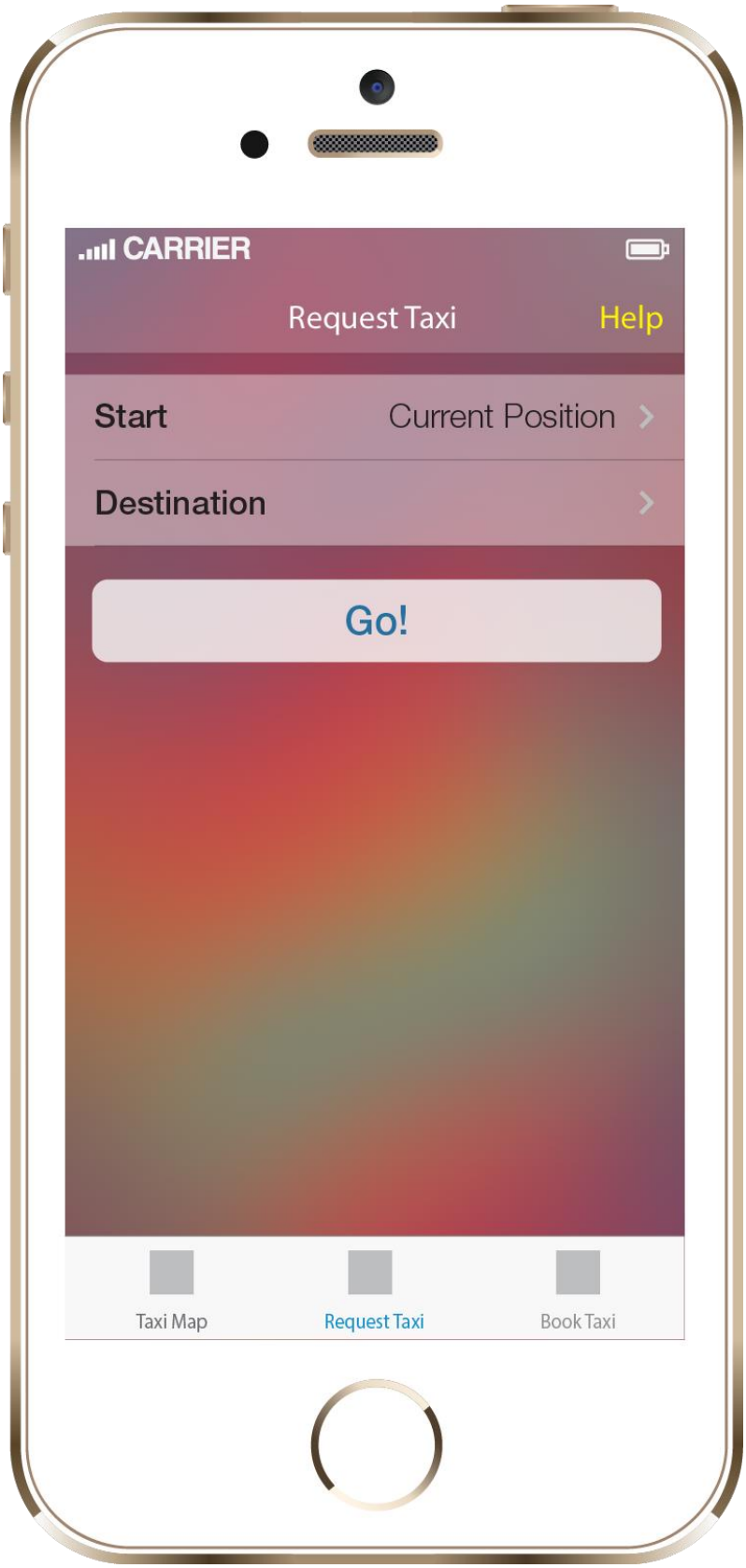
4.2.1.1 Registration form



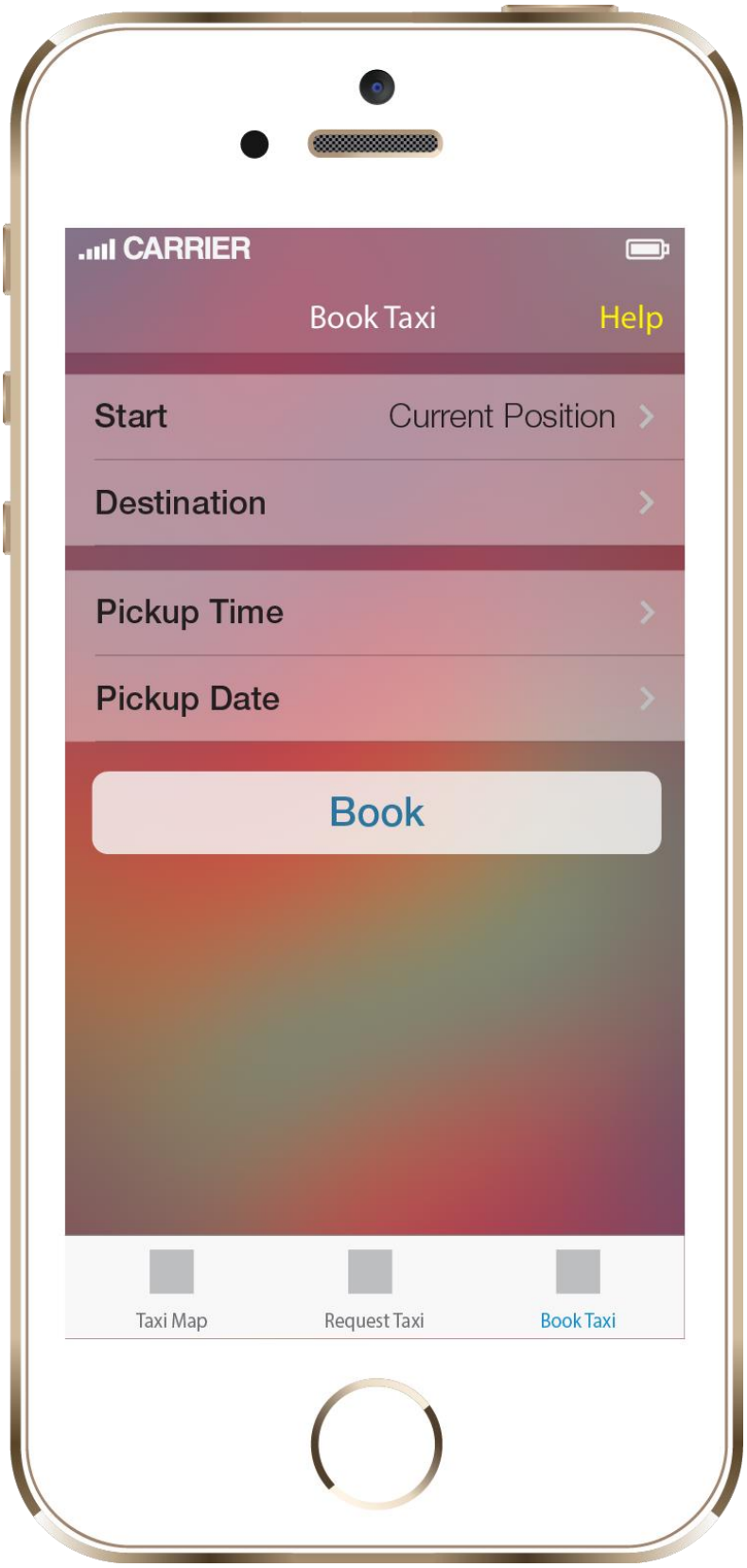
4.2.1.2 Taxi Map



4.2.1.3 Request Taxi

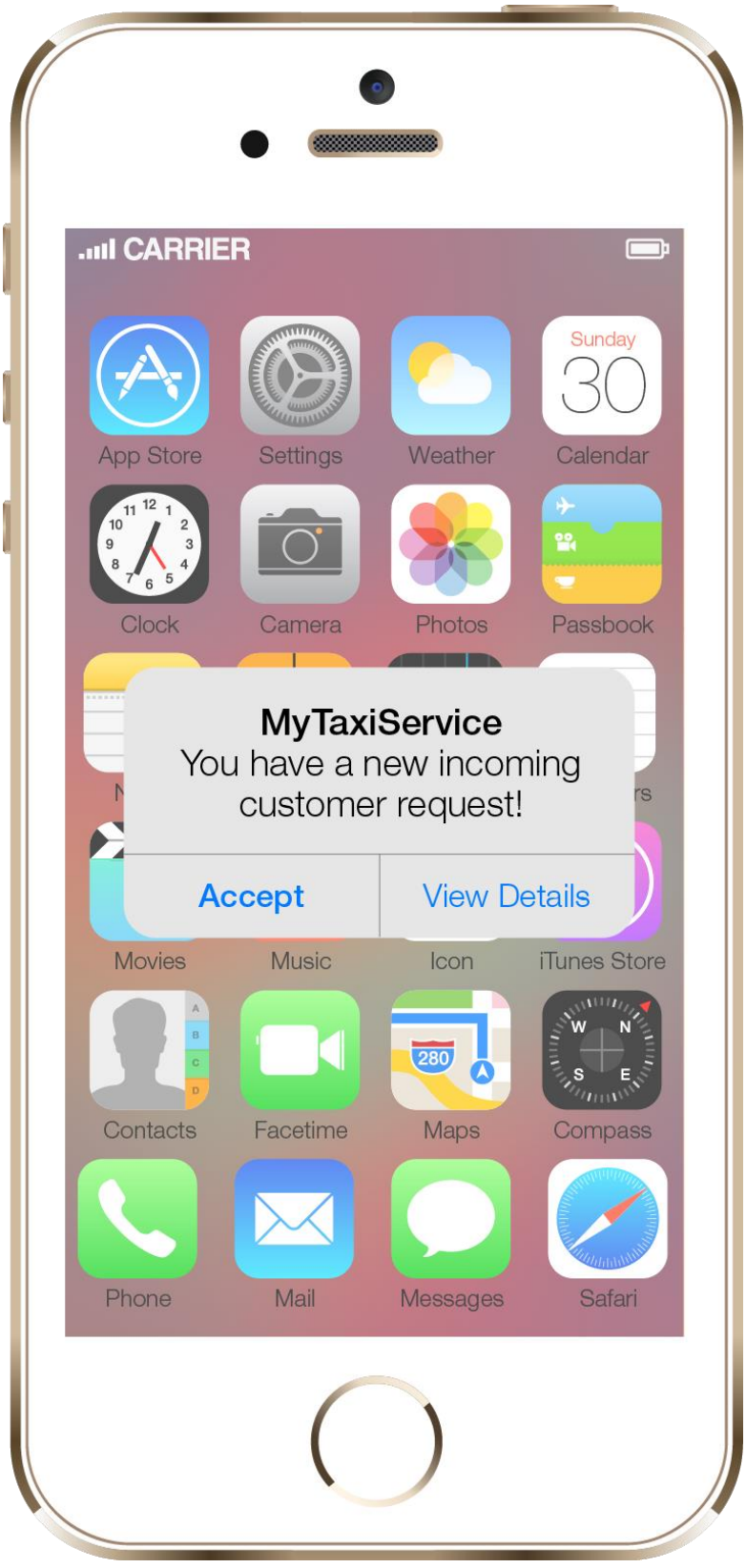


4.2.1.4 Book Taxi

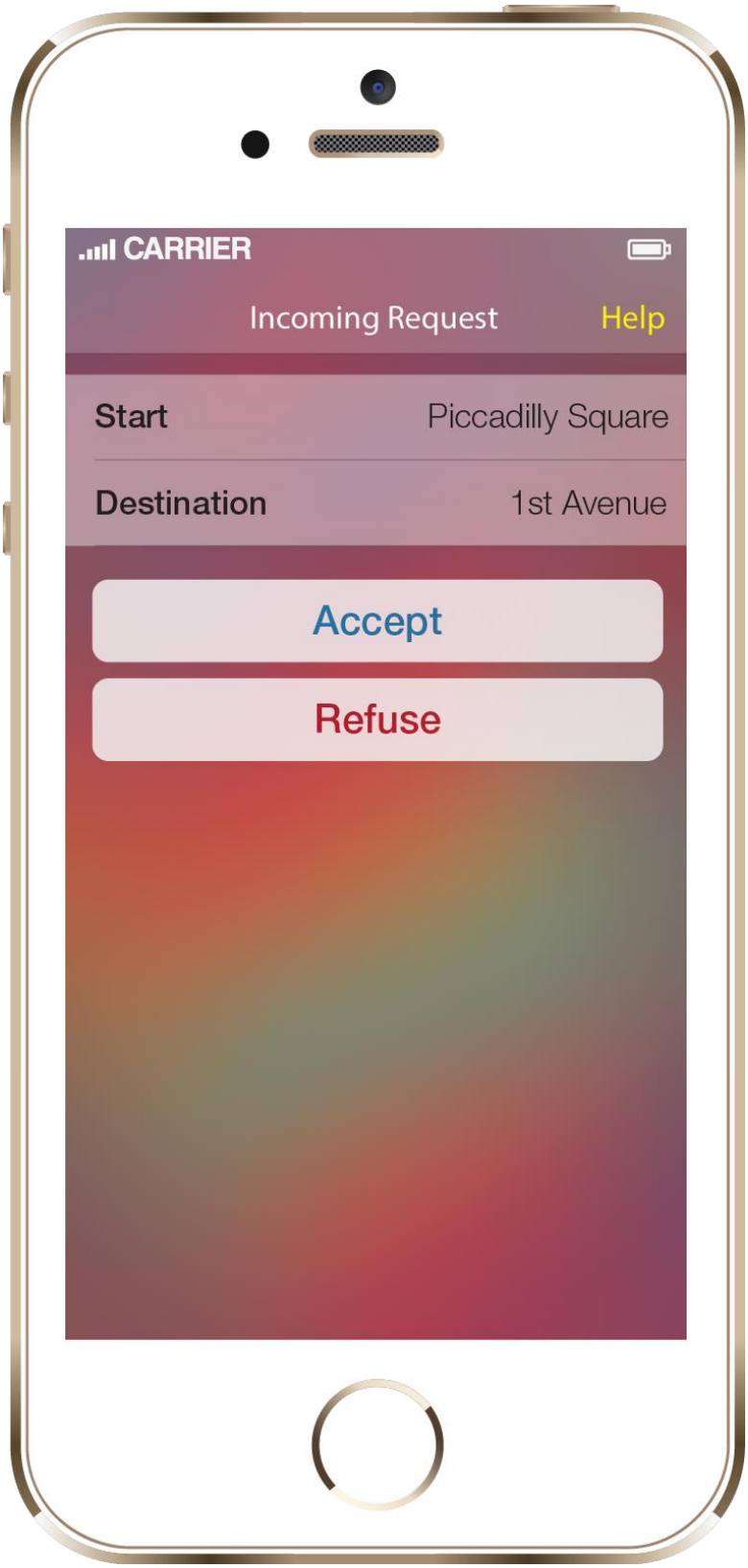


4.2.2 Taxi Driver Mobile App

4.2.2.1 Taxi Request Notification



4.2.2.2 Taxi Request Confirmation



5. Requirements traceability

With respect to what is contained in the RASD, it is possible to conclude that the design decisions explained in this document map well with the products requirements:

- RASD 3.1 (External interface requirements) and RASD 3.2.4.2.R3: the backend modular architecture and the plug-in pattern adoption, which are described in section 2, allows the MyTaxiService software to be extended easily; in particular, as requested in the RASD, it is possible to hook both the user reservation request event and the taxi queue management policy. It is also possible to override the parameters used for taxi selection process through the TaxiSelectionHandler component's plug-in callback.
- RASD 3.5.1 (Ease of use and interface responsiveness): as clarified in section 4 (both with design decisions and UI mockups), we stressed out our products user-friendliness by designing a simple yet complete user interface, which should look familiar to most smartphone users.

6. Appendix

6.1 Hours of work

This is the time spent in order to redact this document:

- Amos Paribocci: 20 hours;
- Lorenzo Pinosa: 22 hours.