Software Engineering 2 Project

Power Enjoy

Niccolo' Raspa
Matteo Marinelli

Text Assumption

- **Park in an Unsafe Area**
  - Restrictive to Prevent this situation from happening
  - One Hour Clock
- **Payments**
  - External Service that takes care payment process
- **Multiple Discounts**
  - Discount only applied if car is a safe area
  - Only Shared Ride discount is cumulative
  - Fines over Discount

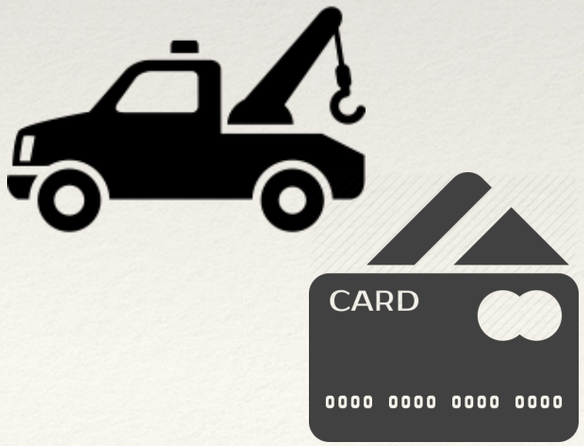
Goals

1. Allows USER to register providing credentials and payment informations
2. Allows USER to login to the system
3. Allows POWER USER to modify its personal informations
4. Allows POWER USER to see locations and battery levels of available CARs
5. Allows POWER USER to reserve one AVAILABLE CAR in a SAFE AREA
6. Allows POWER USER to cancel a CAR RESERVATION
7. Allows POWER USER to unlock his RESERVED CAR when he is close
8. CAR RESERVATION expires after one hour
9. CAR RESERVATION expiration causes 1€ charge to the POWER USER
10. Allows POWER USER to know the current RIDE FEE at any time through
11. The FINAL FEE will be calculated applying discounts/fines according to power enjoy policy
12. POWER USER will be charged of the FINAL FEE after he exits the RESERVED CAR.
13. CARs are automatically locked when parked and the POWER USERS gets out
14. POWER USER with pending payments can’t reserve cars

Domain Assumption

CORRECTNESS AND AVAILABILITY
OF INFORMATIONS

CAR FUNCTIONALITIES



EXTERNAL SERVICES

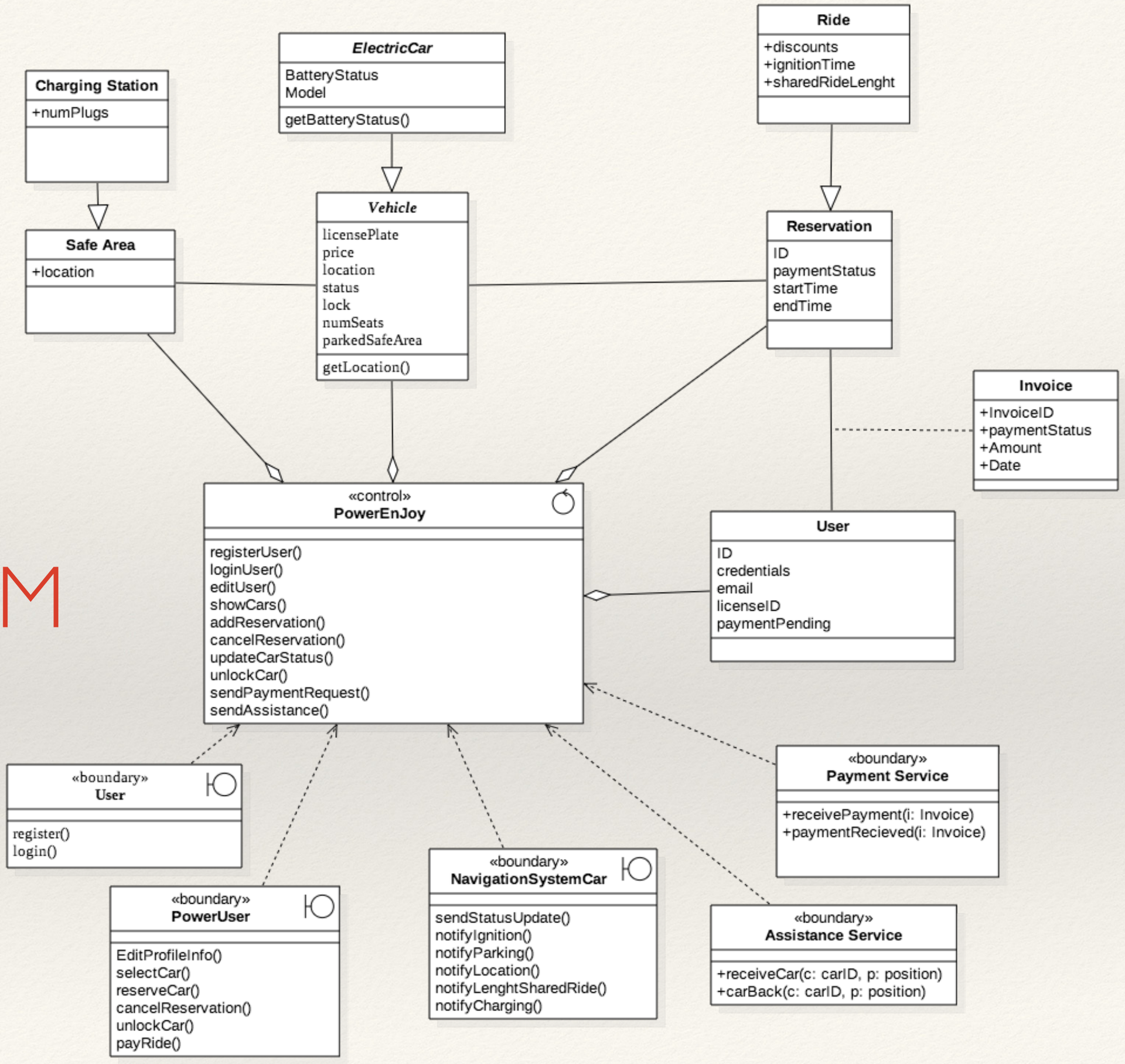
Domain Assumption - Car

- Power Enjoy service employs a particular model of electric car with specific functionalities:
  - **Weight sensors**
  - Ignition sensors
  - Battery Level sensors
  - Global Positioning System (GPS)
  - Automatic keyless entry
  - **Remote control**
  - **Lcd touchscreen**
  - **Internet connectivity**
- Models should also consider this non functional requirements :
  - Battery Length
  - Charging Time
  - Safety
  - Comfort
  - Performance
  - Navigation System

Requirements Derivation

- Scenarios
- Use Cases
- Identification Requirements
- Traceability matrix ensure G = D + R

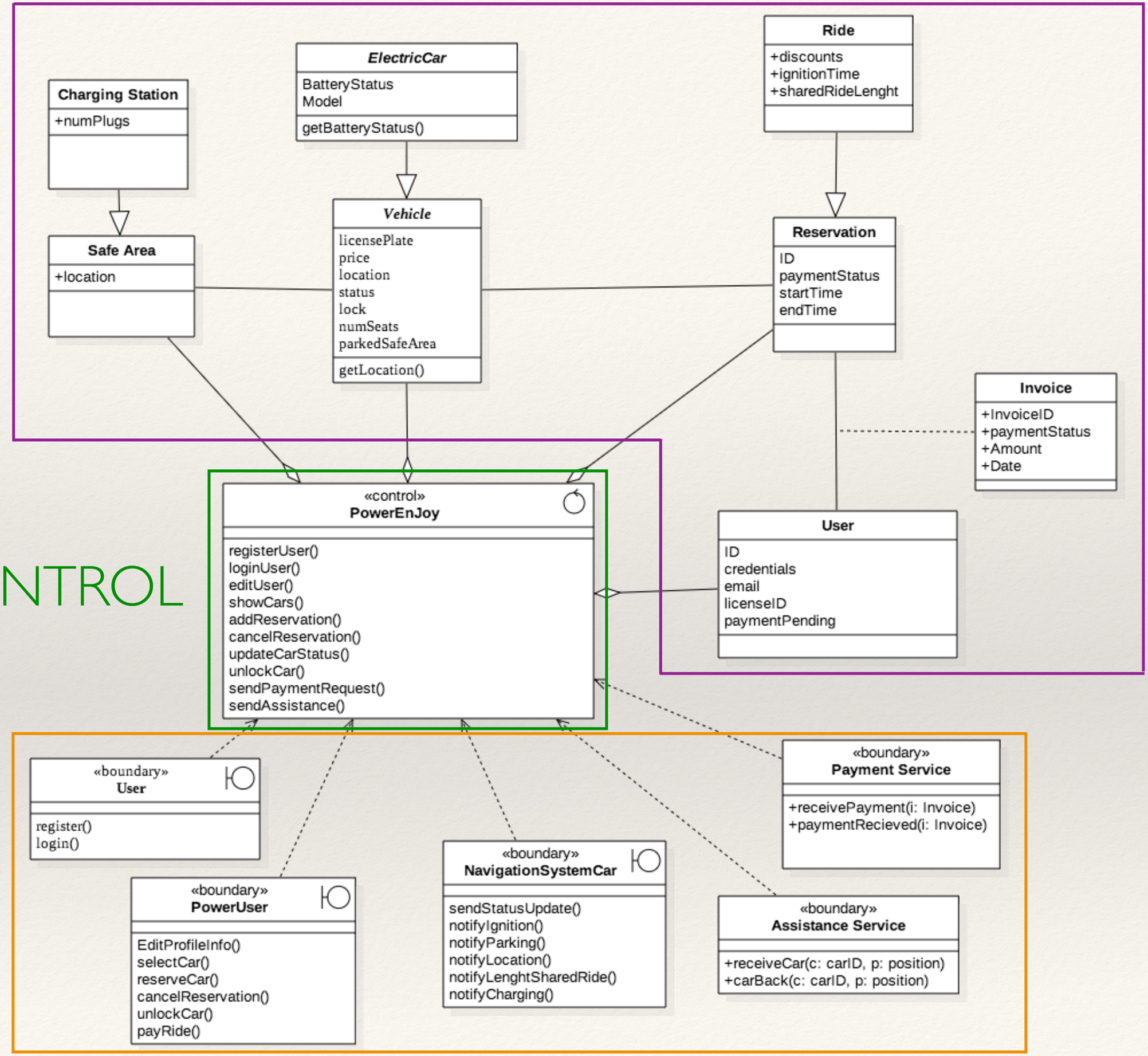
CLASS DIAGRAM



ENTITY

CONTROL

BOUNDARY

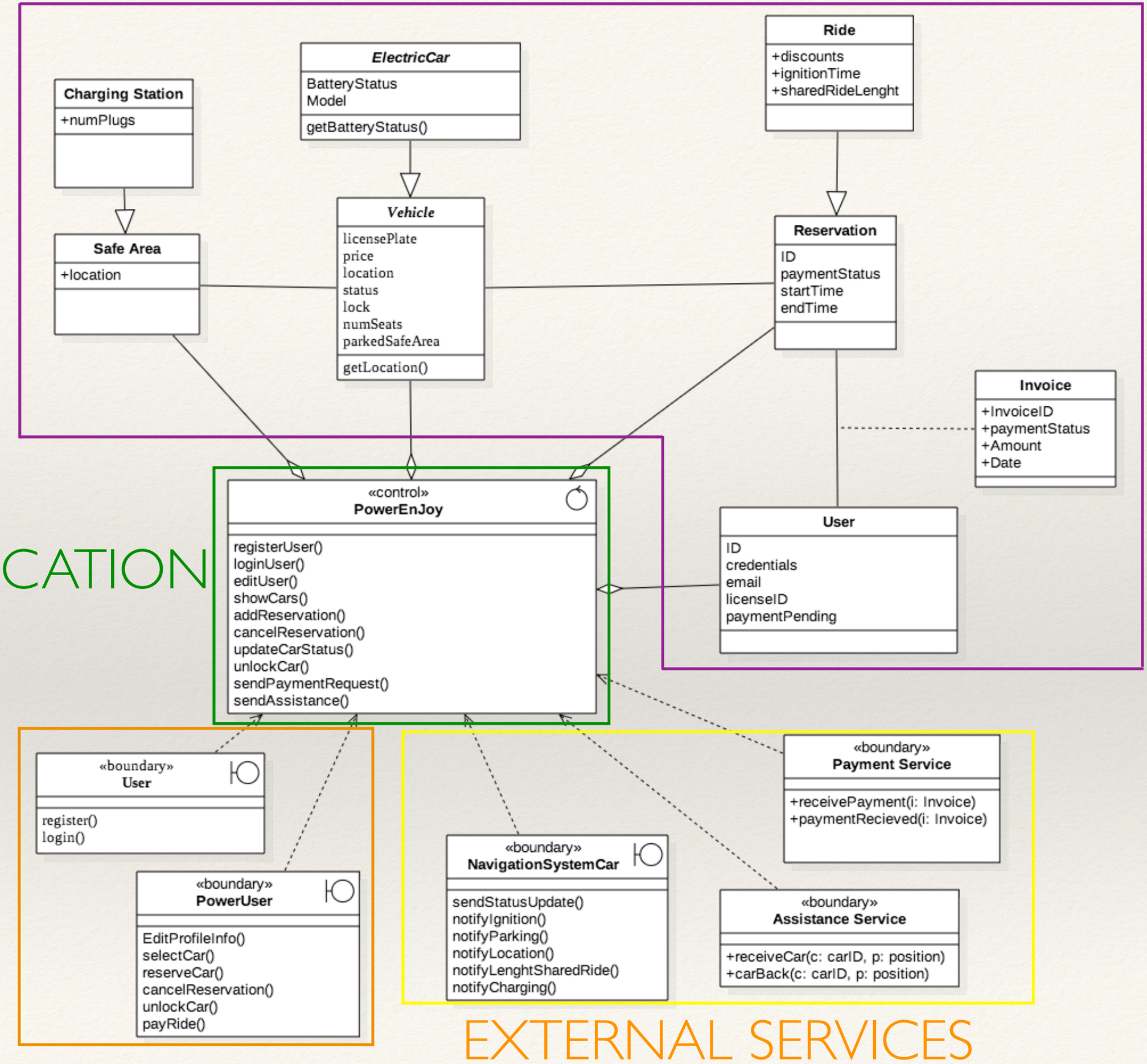


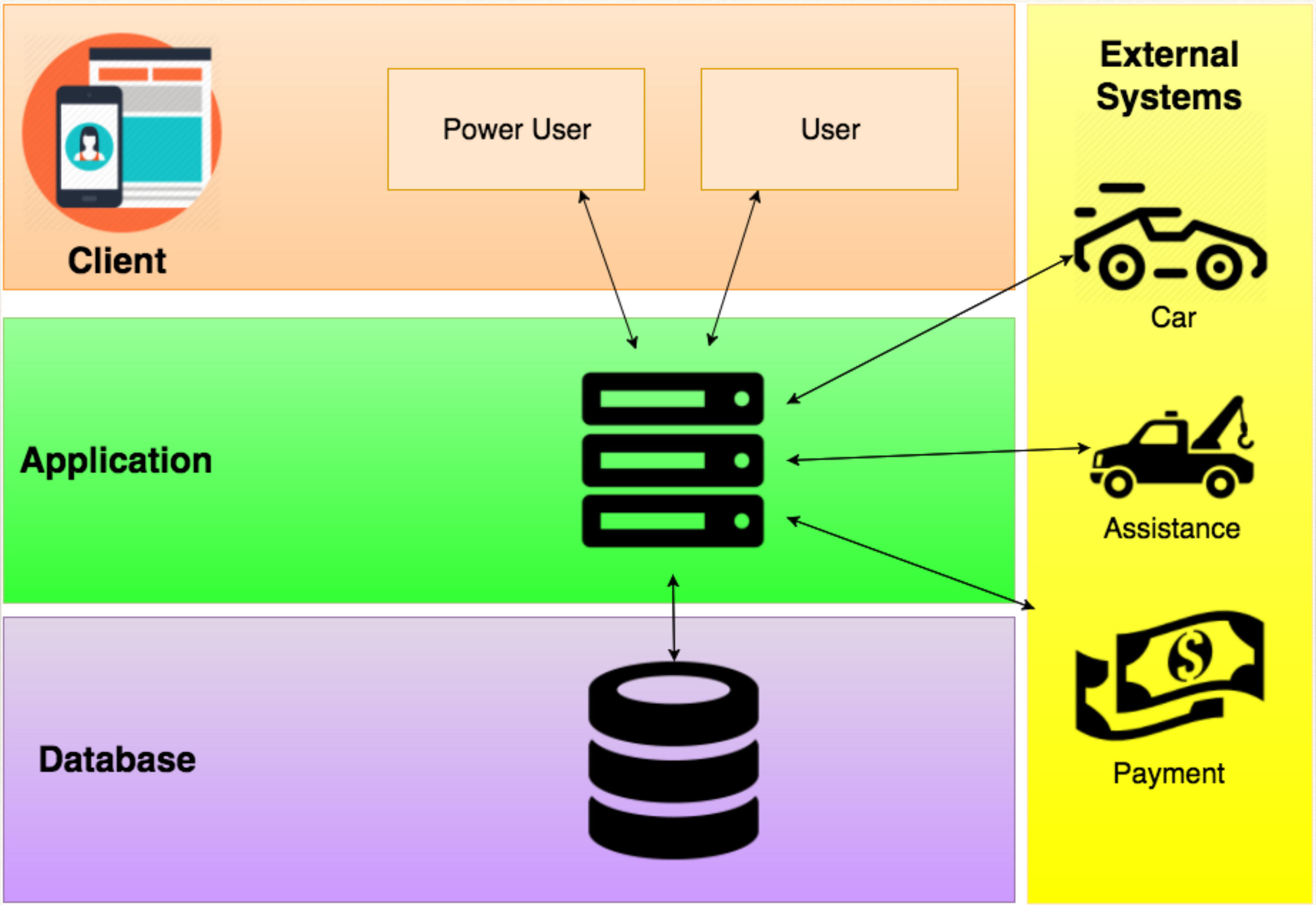
DATABASE

APPLICATION

CLIENT

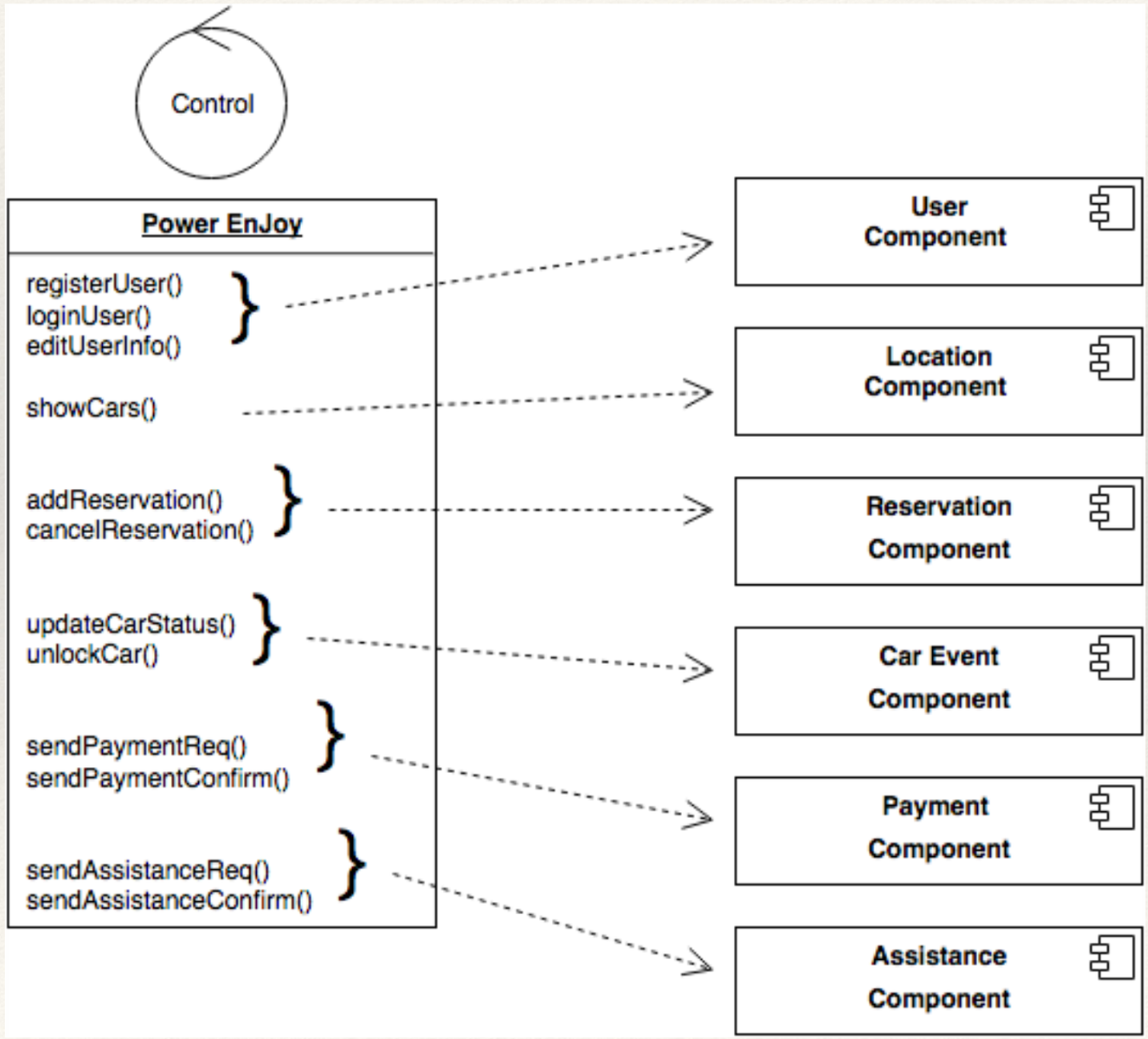
EXTERNAL SERVICES




Client
Power User
User
External Systems
Car
Assistance
Payment
Application
Database

Application Layer

This components implements the logic of the Power Enjoy Application, it's the core of our business



Application Layer - Implementation

- **Java Enterprise Edition 7 (JEE)**
  - Modular Components
  - Large Scale
  - Multi Tiered
  - Scalable

- **Enterprise Java Beans (EJB)**
  - Encapsulate Business Logic

- **GlassFish as Application Server**
  - Supports JEE7
  - Additional Features (Security, Load Balancing)

Client Layer

- **Considerations:**
  - Mobility In Mind
  - Mobile First
- **Expected Functionalities:**
  - Registration
  - Login
  - Edit Profile
  - See Recent Rides
  - Reservation/Ride
  - Make Payment

Client Layer

- **Considerations:**
  - Mobility In Mind
  - Mobile First
- **Expected Functionalities:**
  - Registration
  - Login
  - Edit Profile
  - See Recent Rides

  } Profile Management

  - Reservation/Ride
  - Make Payment

  } Car Sharing

Client Layer

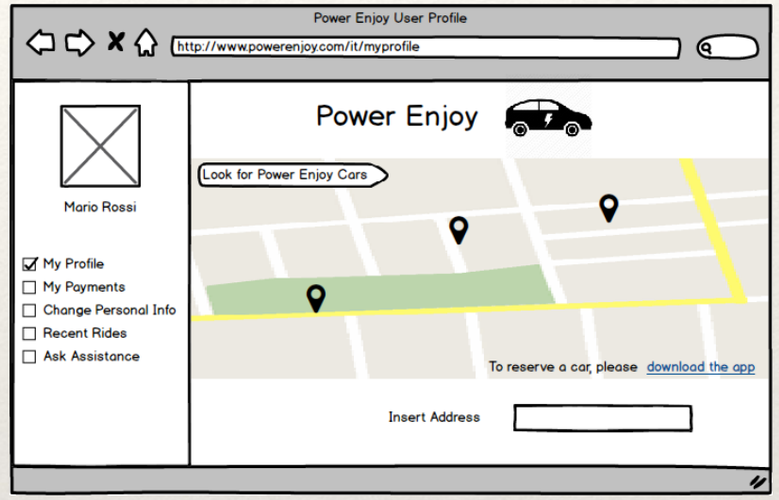
- **Considerations:**
  - Mobility In Mind
  - Mobile First
- **Expected Functionalities:**
  - Registration
  - Login
  - Edit Profile
  - See Recent Rides

  } Profile Management

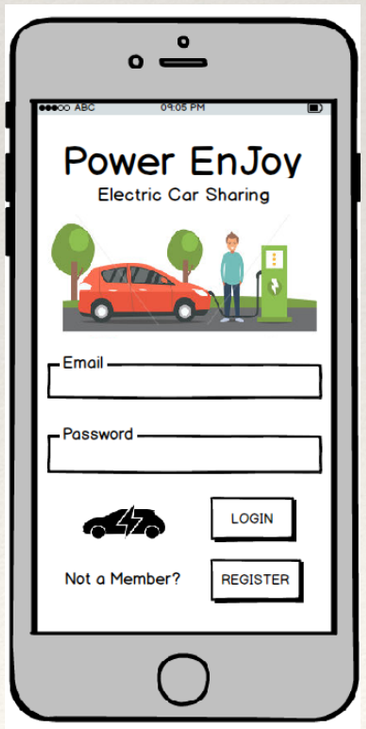
  - Reservation/Ride
  - Make Payment

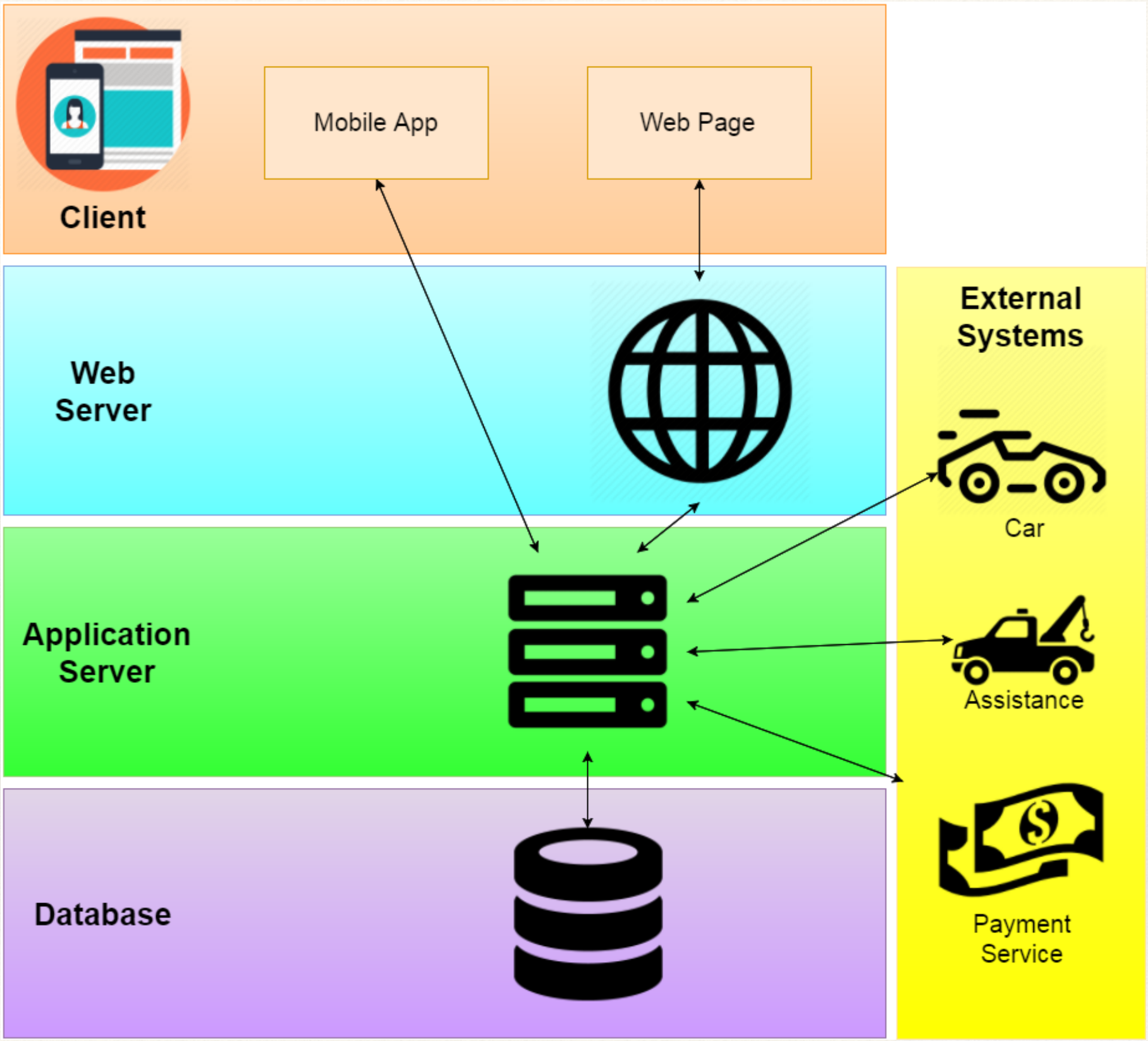
  } Car Sharing

Website



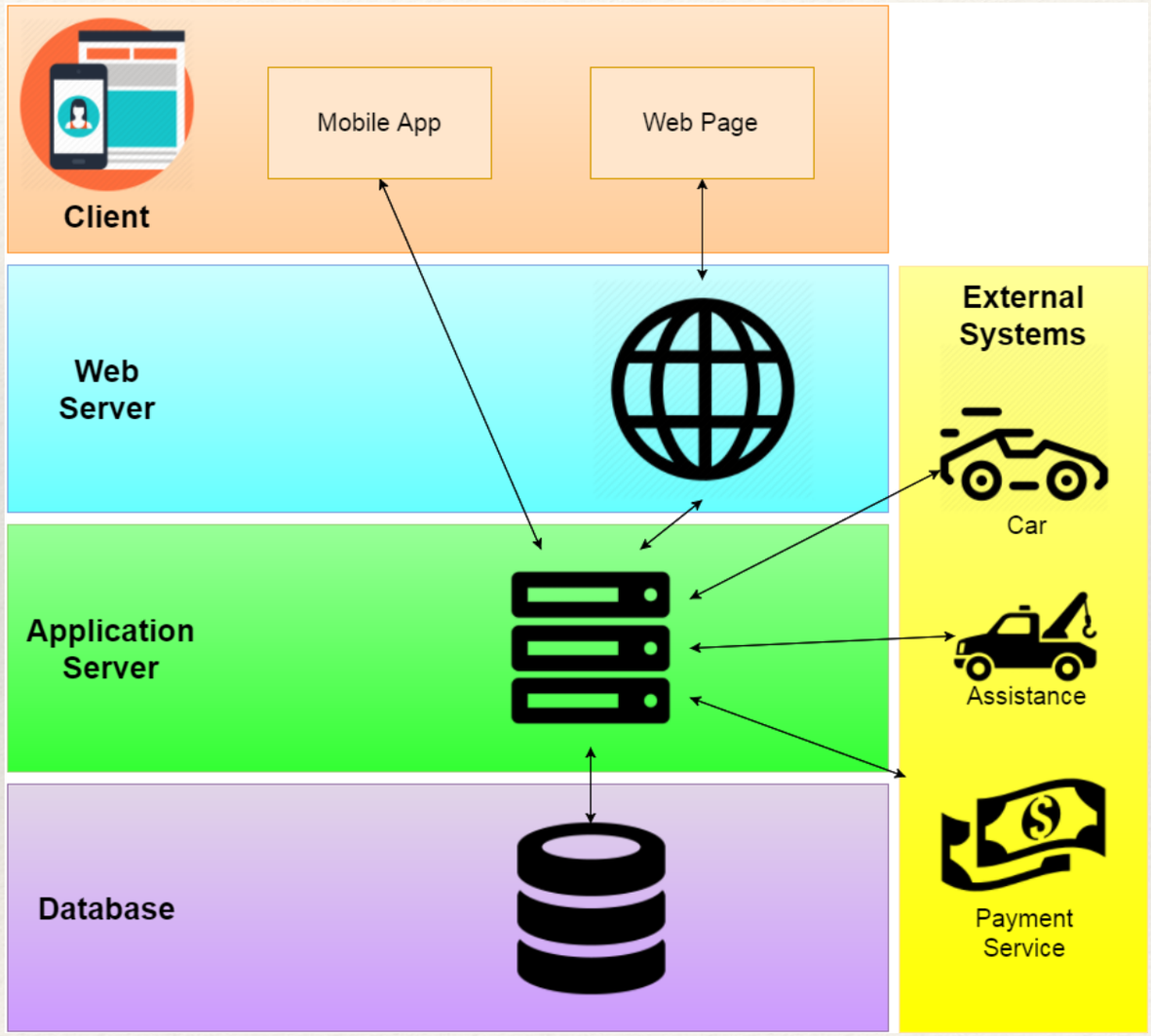
Mobile App




Client
Mobile App
Web Page
Web
Server
External
Systems
Car
Application
Server
Assistance
Database
Payment
Service

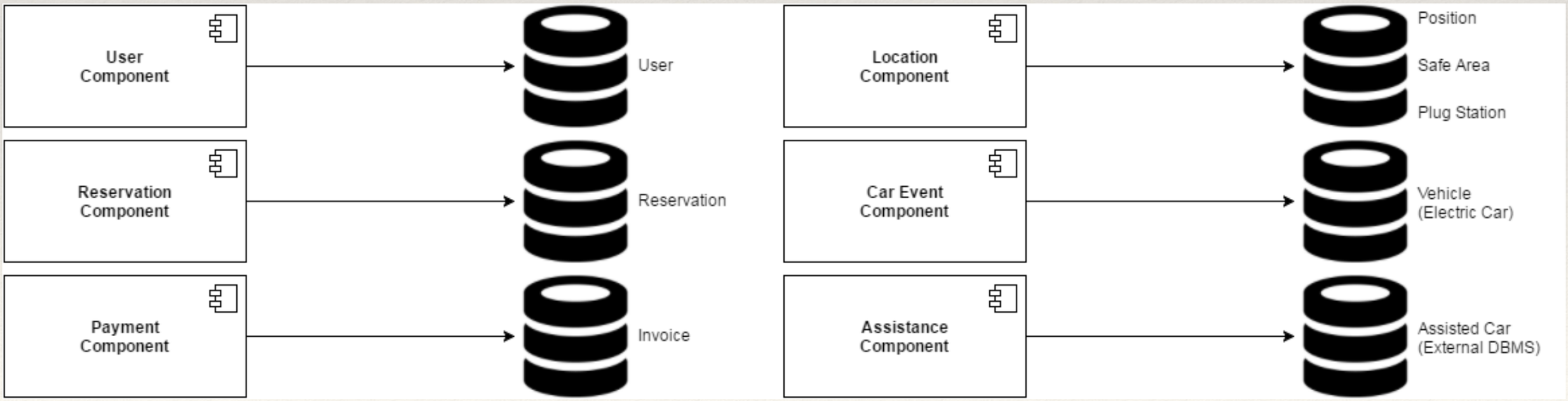
Some Problems…

- **Problems**
- Application Server is the bottleneck of our system
- The performance of this layer is strictly related to the overall performance of the system
- **Solutions**
- Multithreading?
- Sure, but we can do better…



Moving to a SOA approach

- Split the workload among different services
  - Simple and clear Interface to other components
  - Each component is responsible for some entities in the database



Benefits

- It's a more **clean architecture**. Every component implements a service and provides an interface to all the other services.

- Changing/optimising each module **will not affect the whole system** as long as we maintain the same interface for each component.

- It's very flexible, it's will be easy in the future to **add new functionalities**.

- We can **divide the databases among different regions** (e.g. for the city of Milan we don't need to keep track of the cars in Turin)

Web Browser

Mobile App

Web Server

Power Enjoy Application Server

Assistance Service

Assistance System

Payment Service

Payment System

Car Service

Car On Board System

Reservation Service

User Service

Location Service

Database

Integration and Testing

- **Elements to be integrated are:**
  - Integration of the different services inside the Application Layer
  - Integration of different tiers (Client - Web - Application)
  - Integration and configuration with third party systems (Payment System, Assistance System)

Integration Strategy

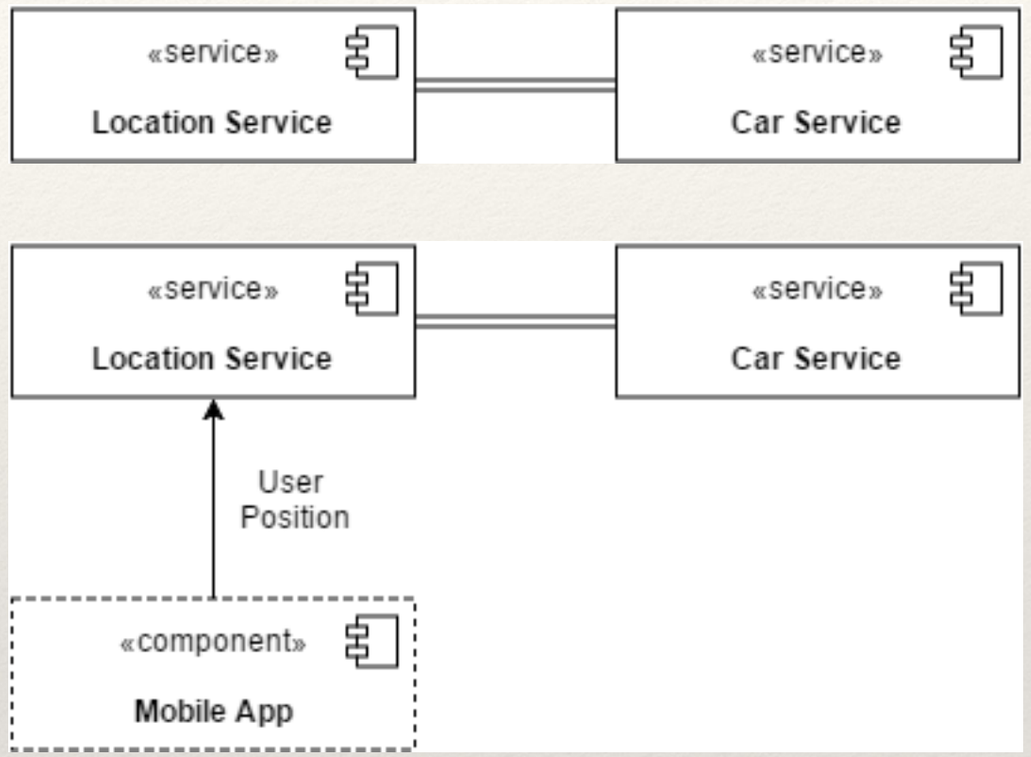
- **Mixture of the bottom-up and functional-grouping**
  - critical components first
  - start from small independent service
  - group them to implement complex functionalities
- **Relation with third party**
  - fixes might delay the process

Integration Strategy

1. Ensure that services in relations with external systems works as expected
2. Ensure that we have control over the Car
3. Integration of Services
4. Integration with top layers
5. Alpha Test

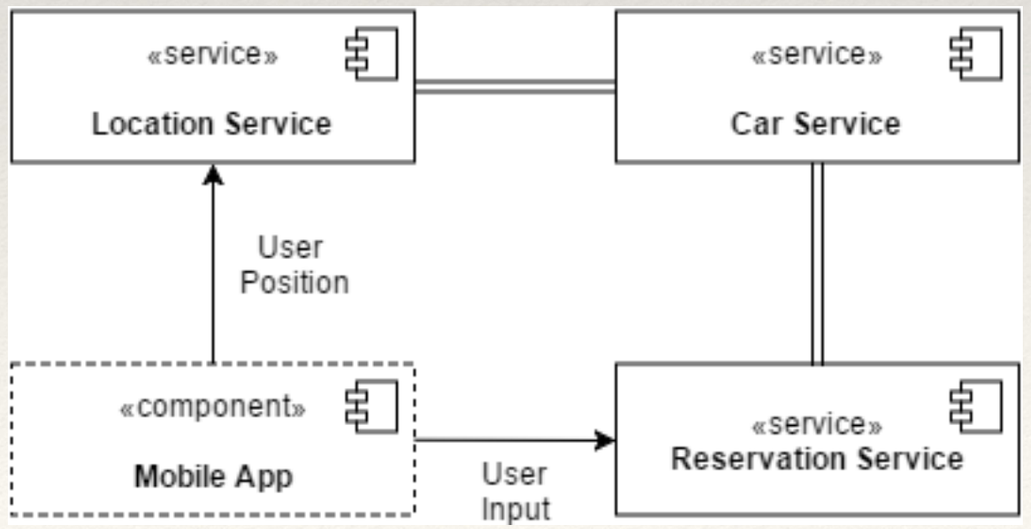
Integration of Services

LOCATION OF VEHICLES



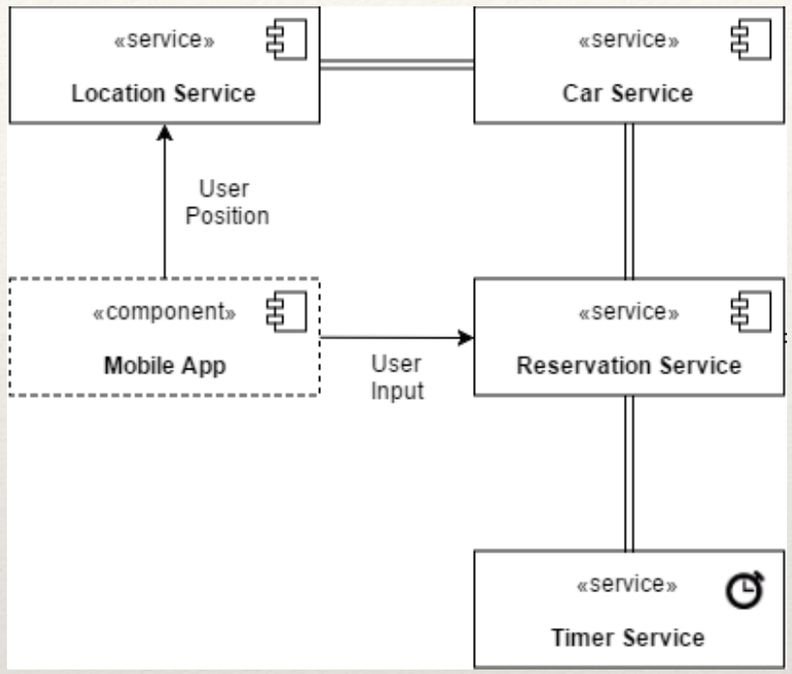
LOCATION OF USER

RESERVATION

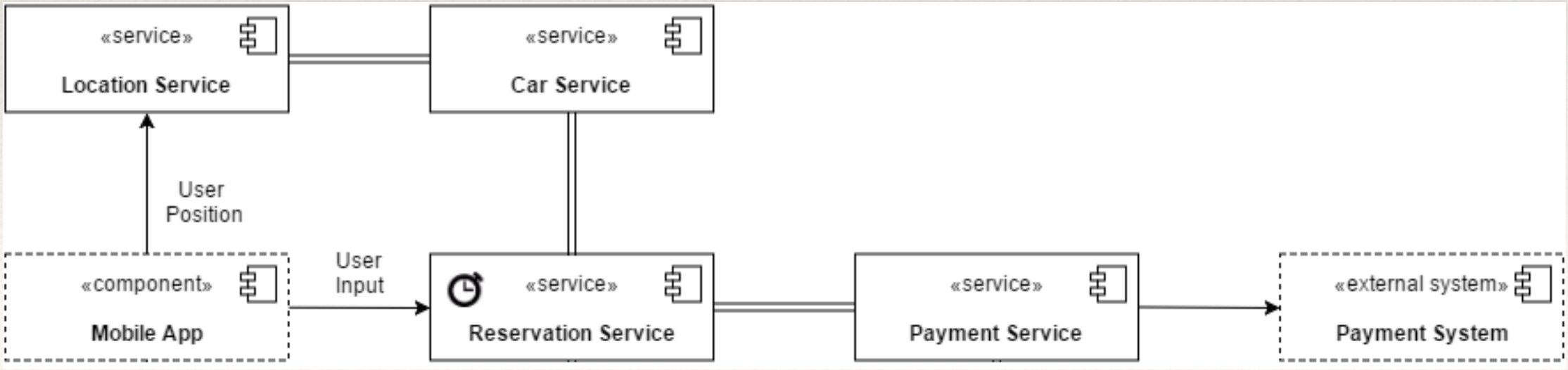


Integration of Services

TIMING

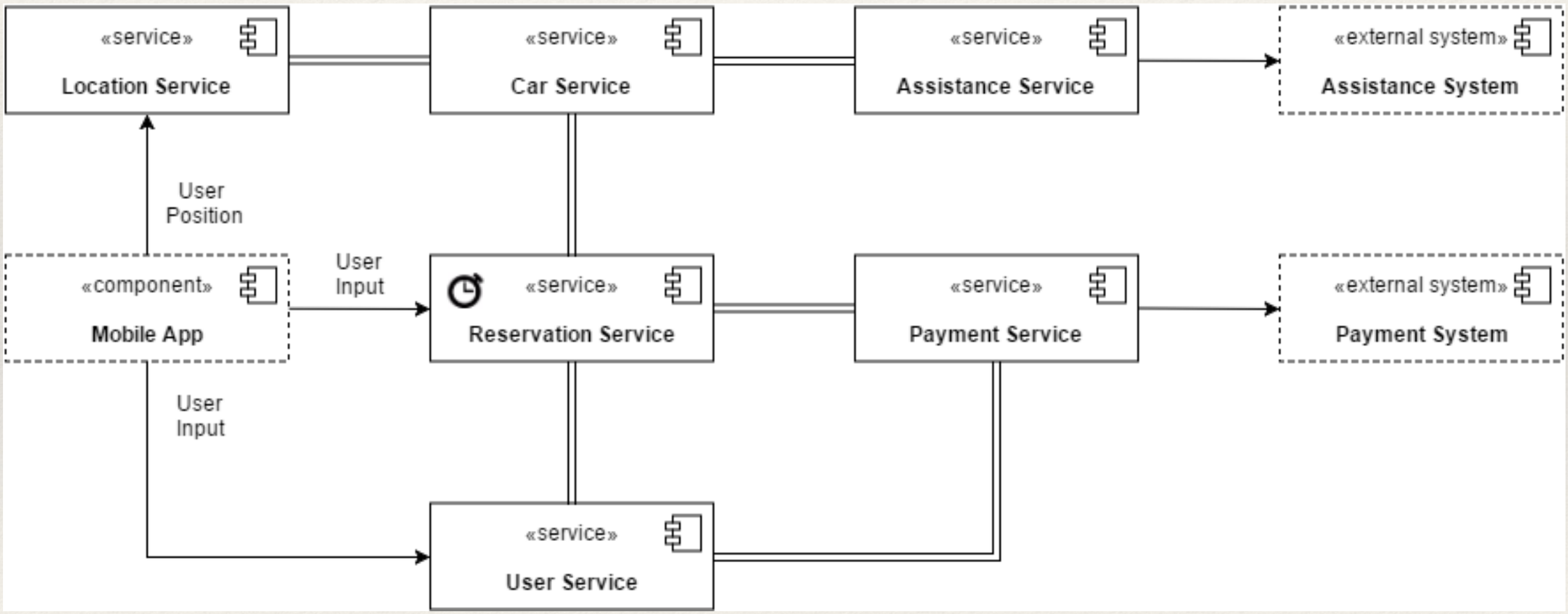


PAYMENT



Integration of Services

WHOLE SYSTEM



Tools Used

- Mockito
- Arquilian
- JUnit
- Manual Testing

Project Plan

- Beginning of the project → Early Design Approach
- Function Points and Drivers (COCOMO) values based on our real skill and experience
- $SLOC_{Avg}$ = 7728 lines of code
- Effort = 31,47 PM
- Duration ≈ 16 Months with 2 developers
  - High value but reasonable since Early Design approach

Scheduling


Research & PP
SEP 2016
RASD
OCT 2016
DD
NOV 2016
Matteo
Customer Analysis
FPs Estimation
Cost/Benefits Analysis
COCOMO II Estimation
Domain, Goals
Use Cases
Functional Requirements
Class Diagram
Component View
Runtime View
System Architecture
Requirements Traceability
Niccolo'
Market Analysis
Scheduling
Resource Allocation
Feasibility Study
Functional Requirements
Non Functional Requirements
World & Machine
UML Diagrams
System Architecture
UI Mockups
Deployment View
High Level Components
MILESTONE
MILESTONE
ITPD
DEC 2016 - JAN 2017
Implementation
JAN 2017 - SEP 2017
Integration
SEPT 2017 - DEC 2017
Matteo
Driver & Stubs Required
Test Cases
Individual Test Description
Single Unit Coding
Code Inspection
Single Unit Testing
Subsystems Integrations
Alpha Test
Deployment
Niccolo'
Integration Strategy
Test Data & Equiment
Individual Test Description
Single Unit Coding
Code Inspection
Single Unit Testing
Components Integration
Alpha Test
Deployment
MILESTONE
MILESTONE

Poject Risks

- Changing Requirements
  - Why? Most recurrent and also unpredictable
  - Strategy: Traceability Information and information hiding in design
- Personnel Shortfall
  - Why? Just 2 developers 1 fall = +50% delay
  - Strategy: Positive Work Environment

Technical Risks

- Defecting Components of the Car System
  - Why? Delays in the Testing Stage
  - Strategy: Present partial developement
- Software too difficult to use for avarage costumers
  - Why? Important point for the Success or Failure of the product
  - Strategy: Revisit UI and UX
- Also others…

Questions