



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. Allow USER to register providing credentials and payment informations
2. Allow USER to login to the system
3. Allow POWER USER to modify its personal informations
4. Allow POWER USER to see locations and battery levels of available CARs
5. Allow POWER USER to reserve one AVAILABLE CAR in a SAFE AREA
6. Allow POWER USER to cancel a CAR RESERVATION
7. Allow 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. Allow 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

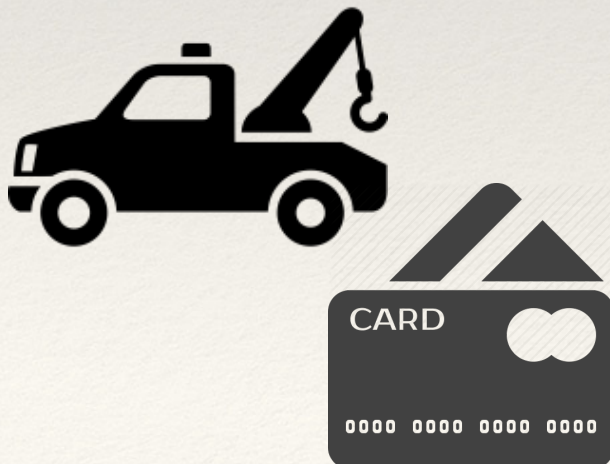
Domain Assumption



CORRECTNESS AND AVAILABILITY
OF INFORMATIONS



CAR FUNCTIONALITIES

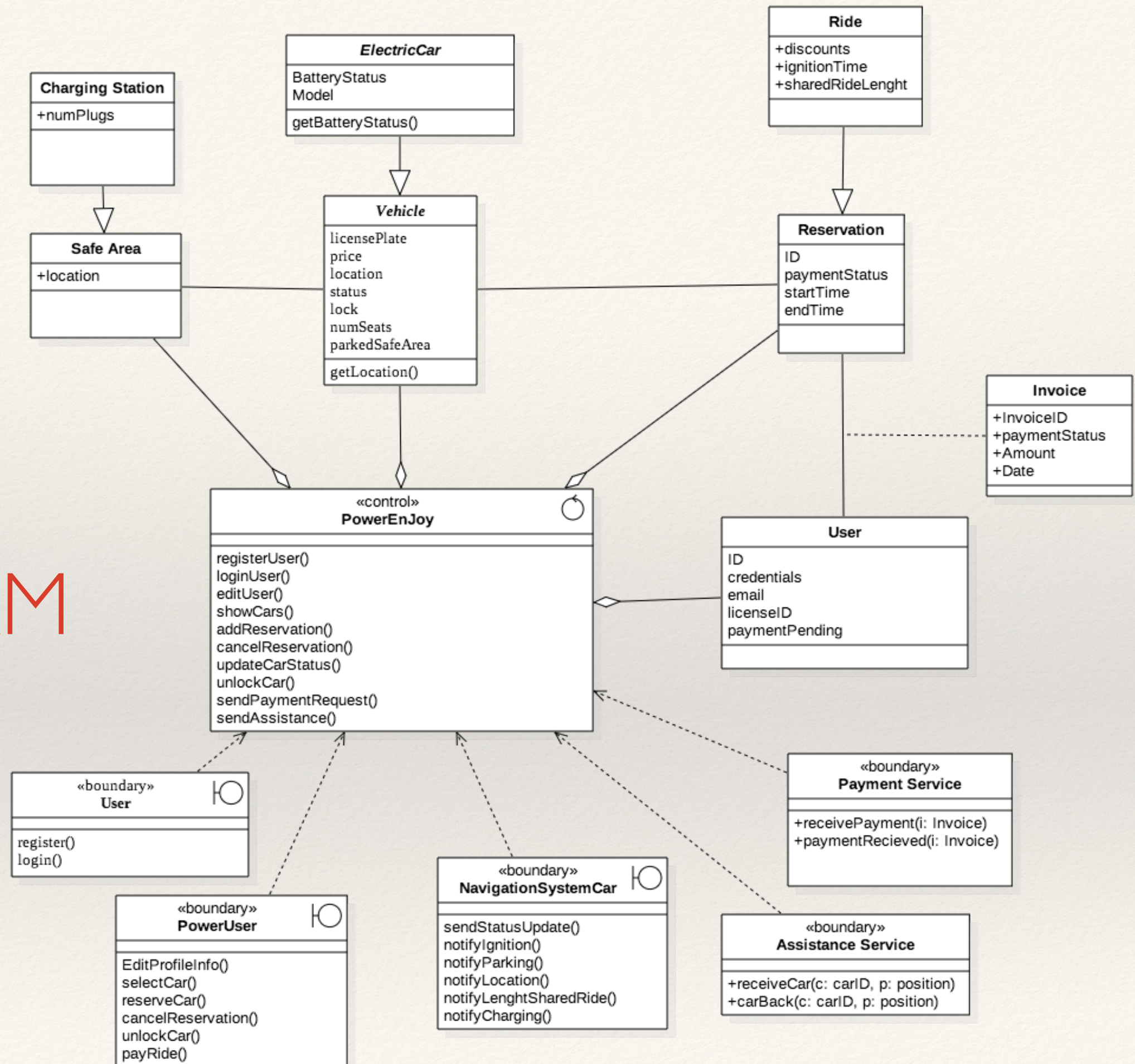


EXTERNAL SERVICES

Requirements Derivation

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

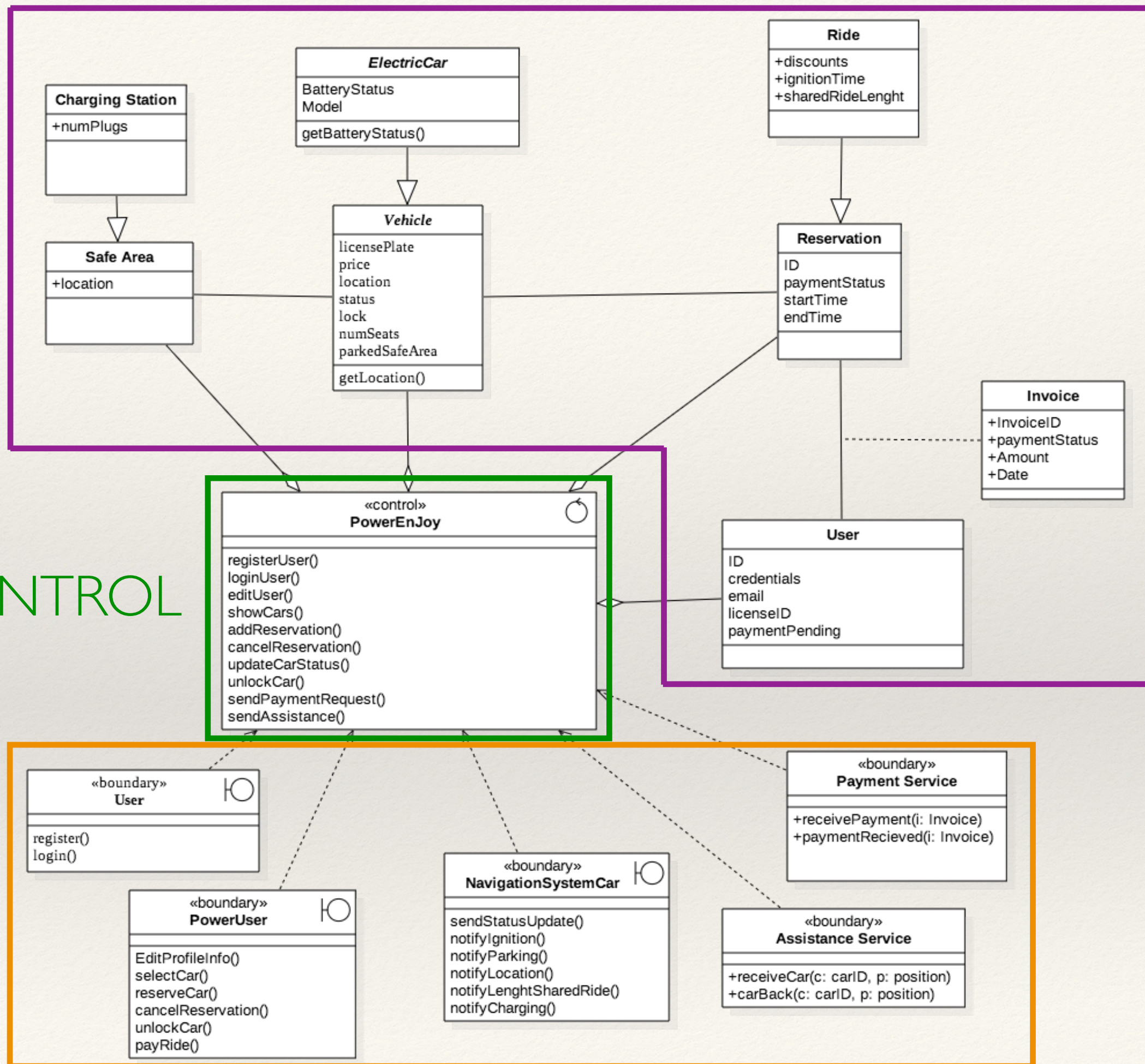
CLASS DIAGRAM



ENTITY

CONTROL

BOUNDARY

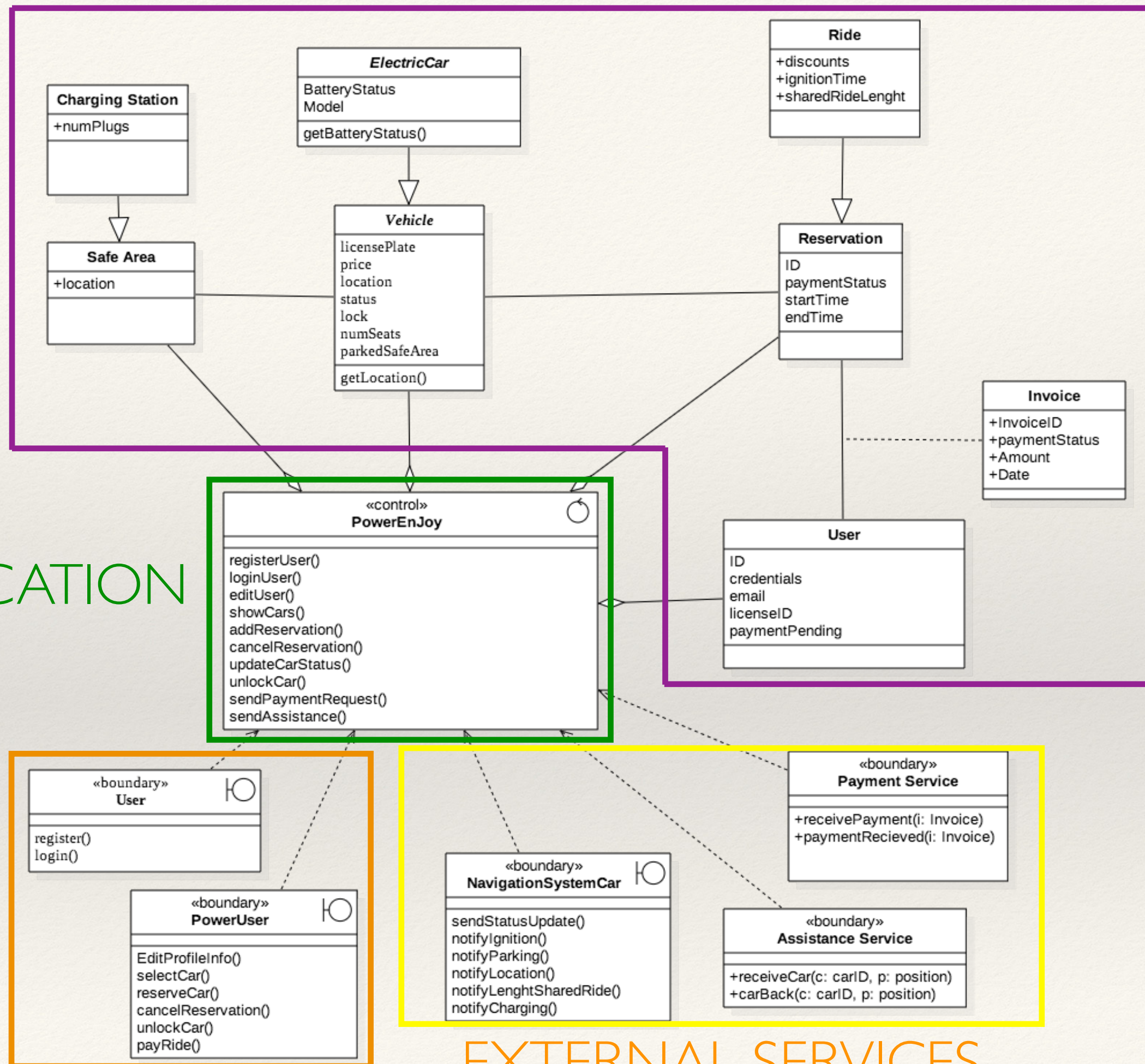


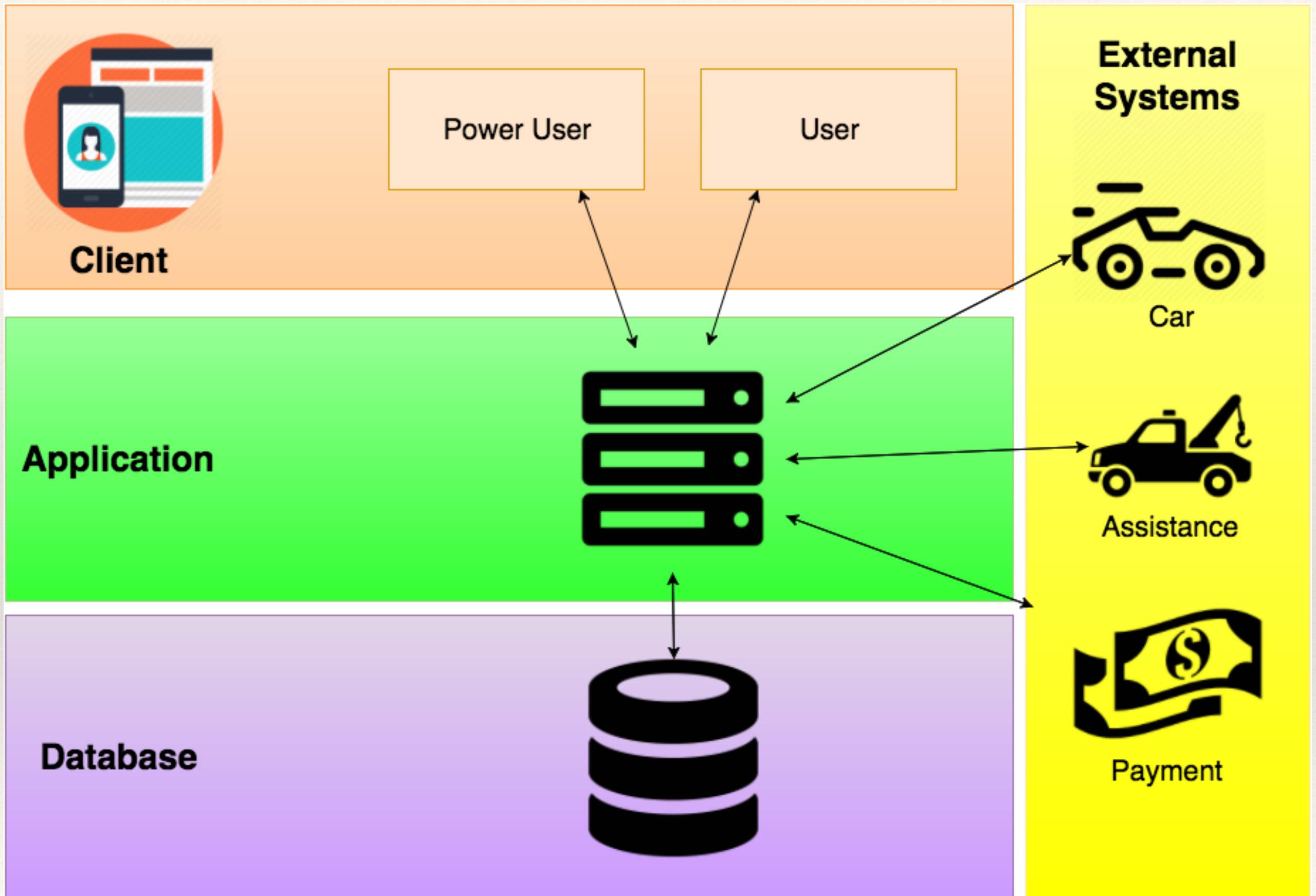
DATABASE

APPLICATION

CLIENT

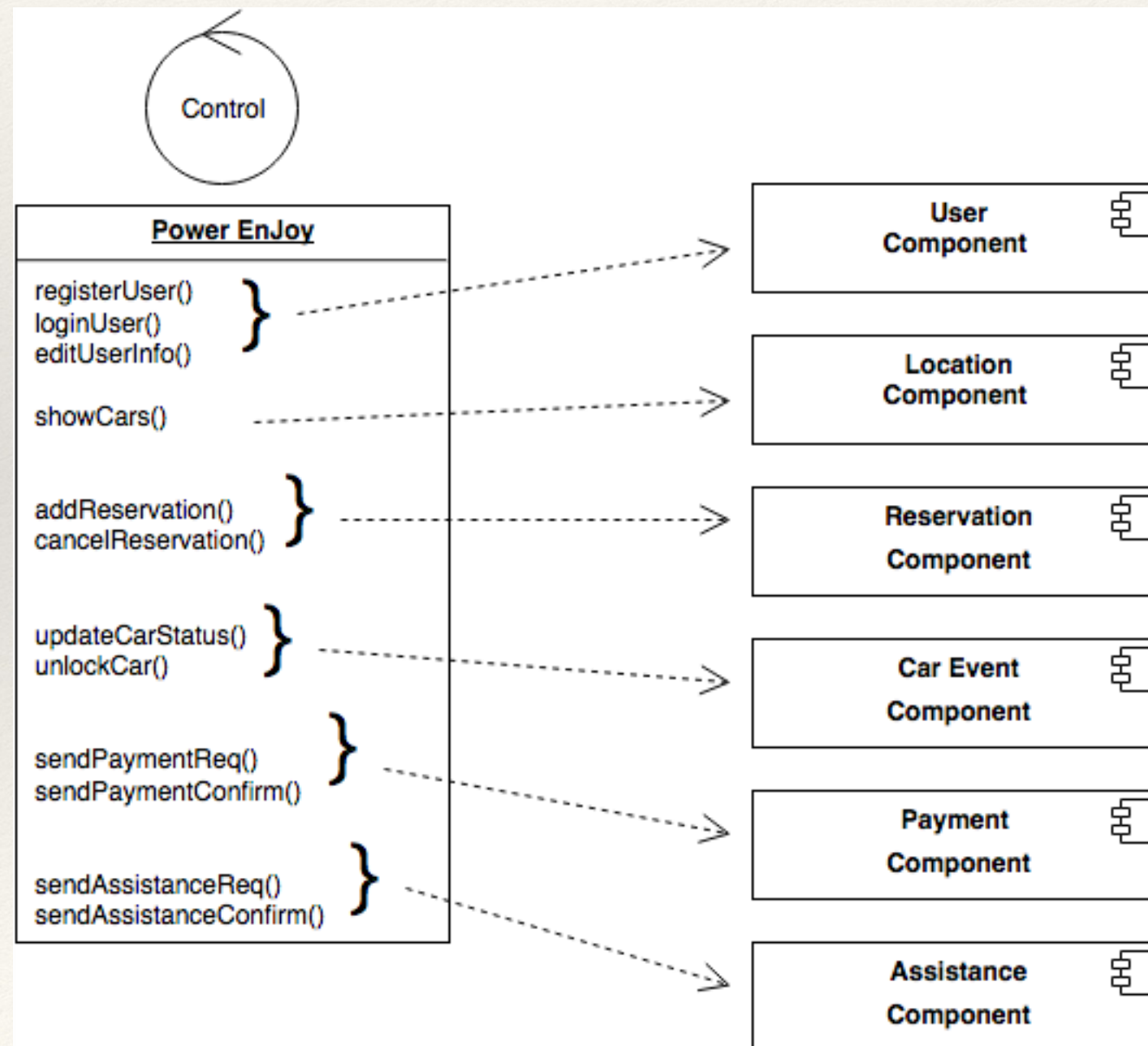
EXTERNAL SERVICES





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

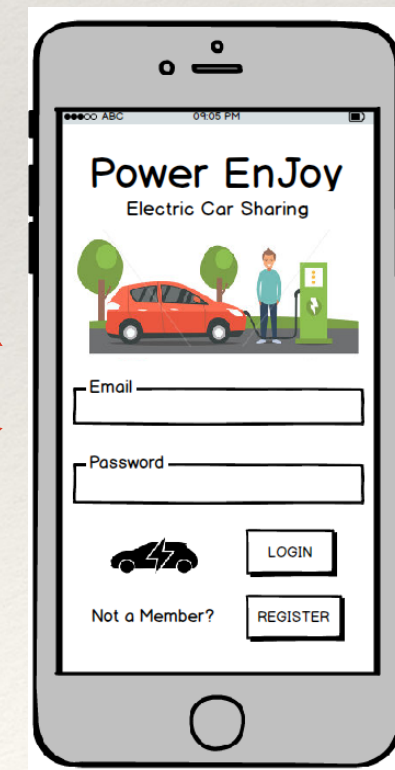
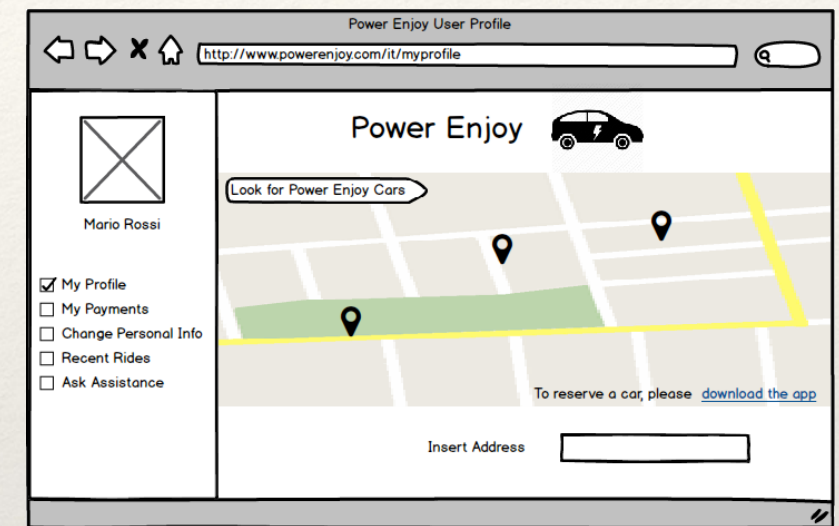
❖ Expected Functionalities:

- ❖ Registration
- ❖ Login
- ❖ Edit Profile
- ❖ See Recent Rides
- ❖ Reservation/Ride
- ❖ Make Payment

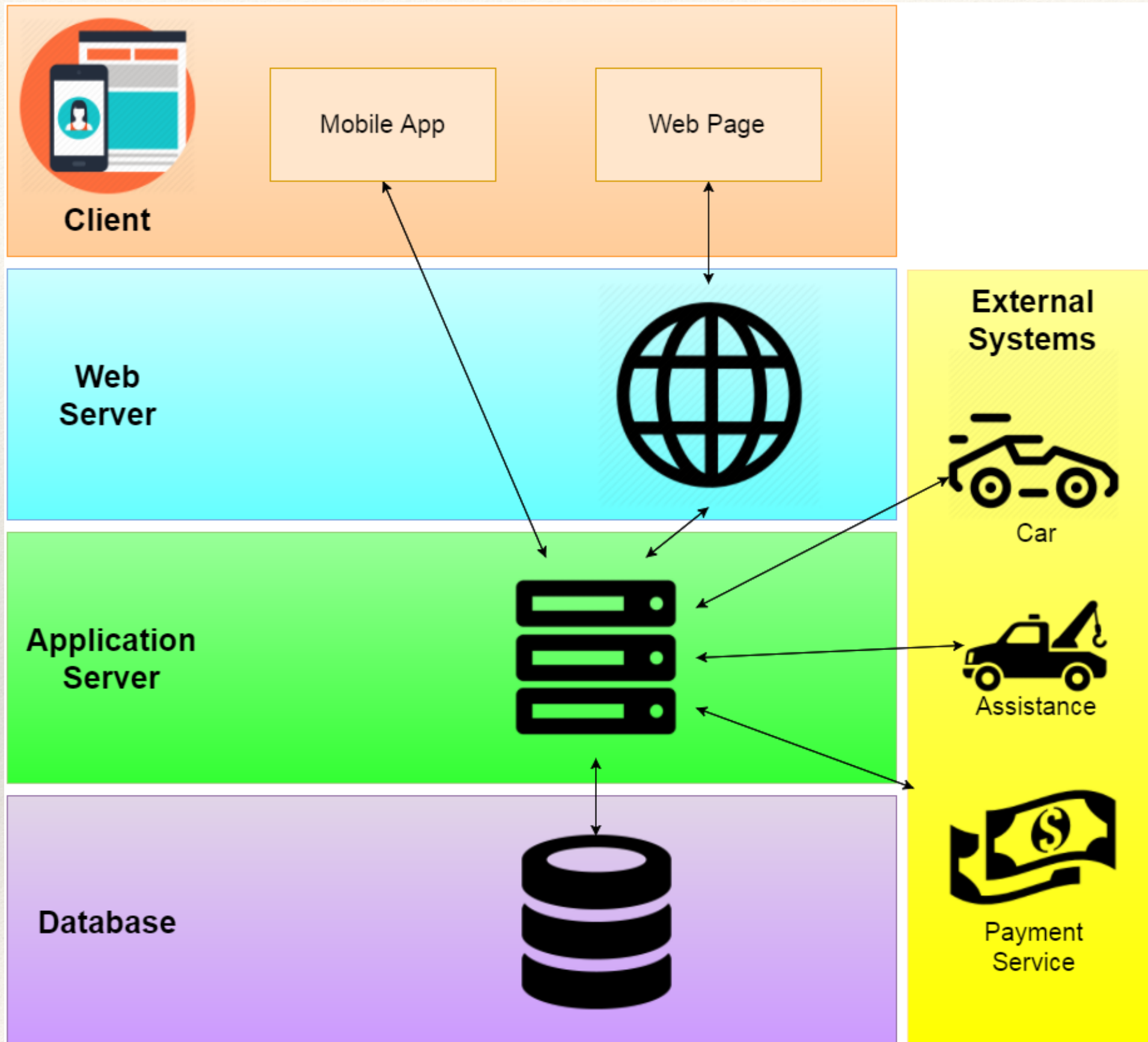
Profile Management

Car Sharing

Website



Mobile App



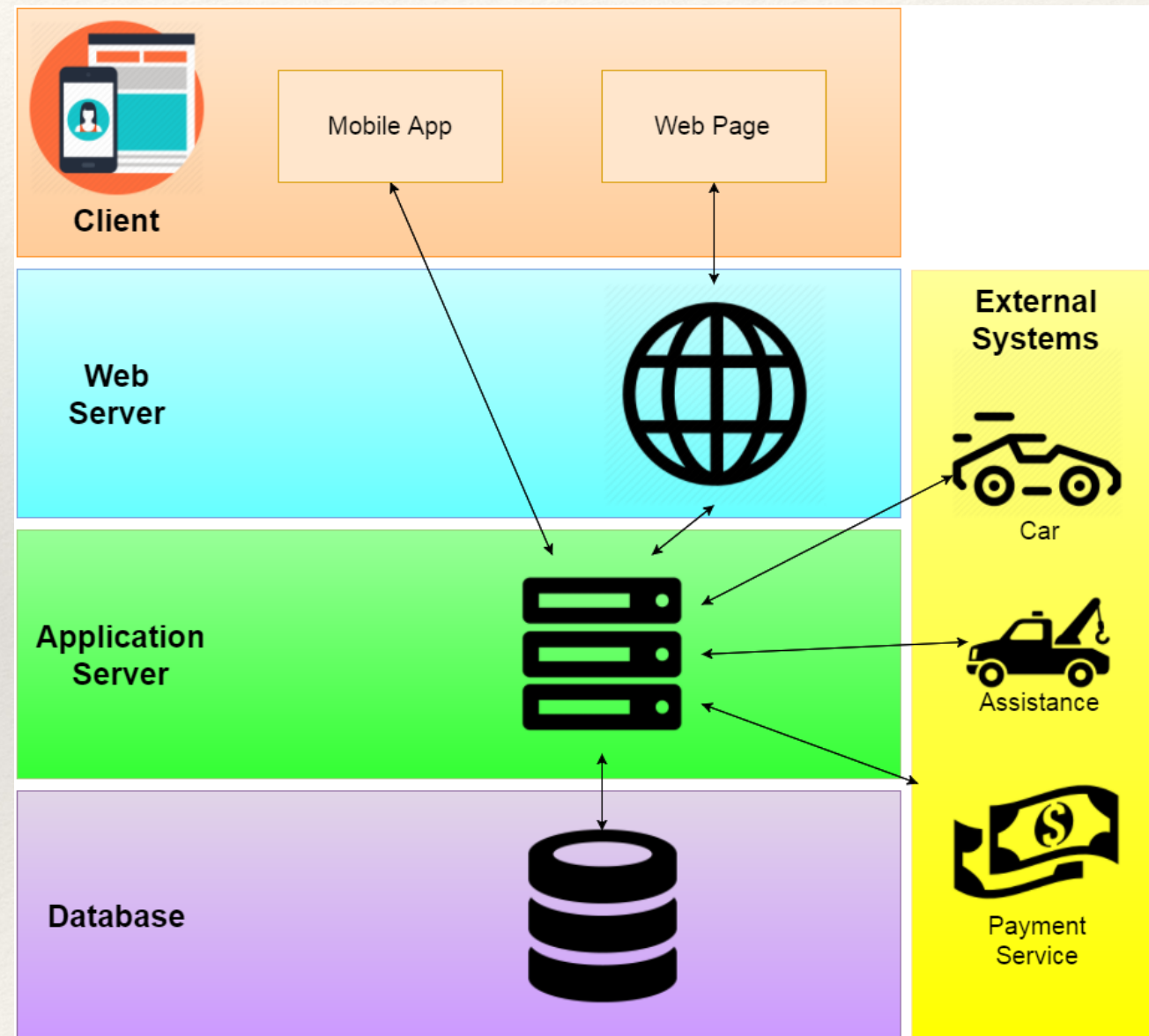
Some Problems...

❖ Problems

- ❖ Application Server is the bottleneck of our system
- ❖ The performance of this layer is strictly related to the overall performance of the system

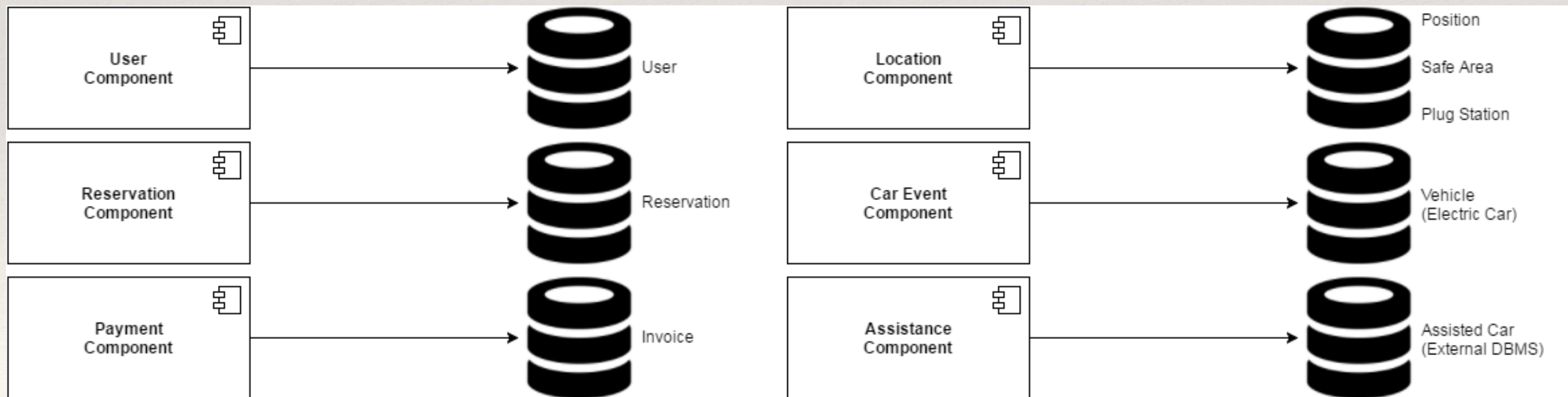
❖ Solutions

- ❖ Multithreading?
- ❖ Sure, but we can do better...



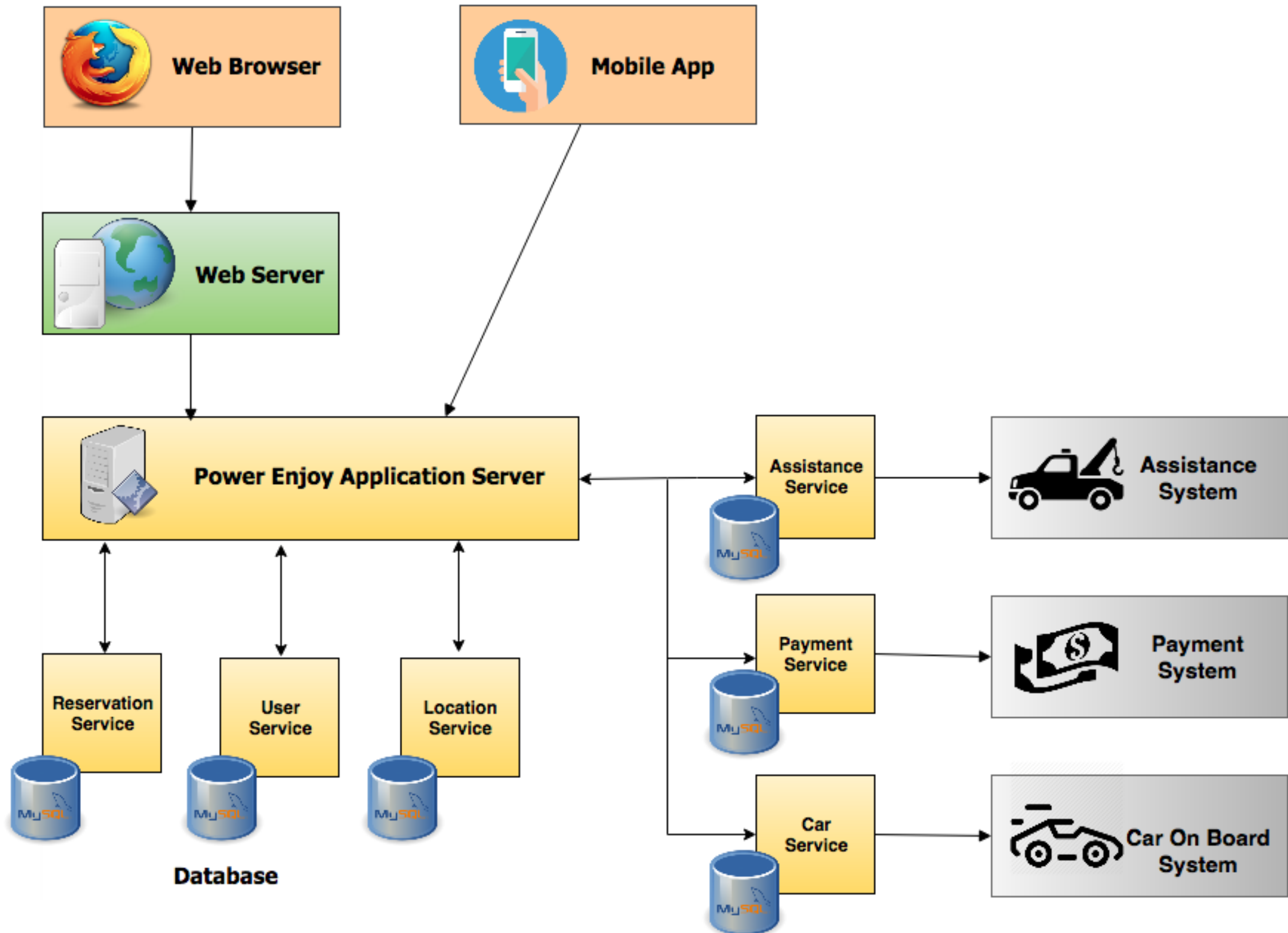
Moving to a SOA approach

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



Benefits

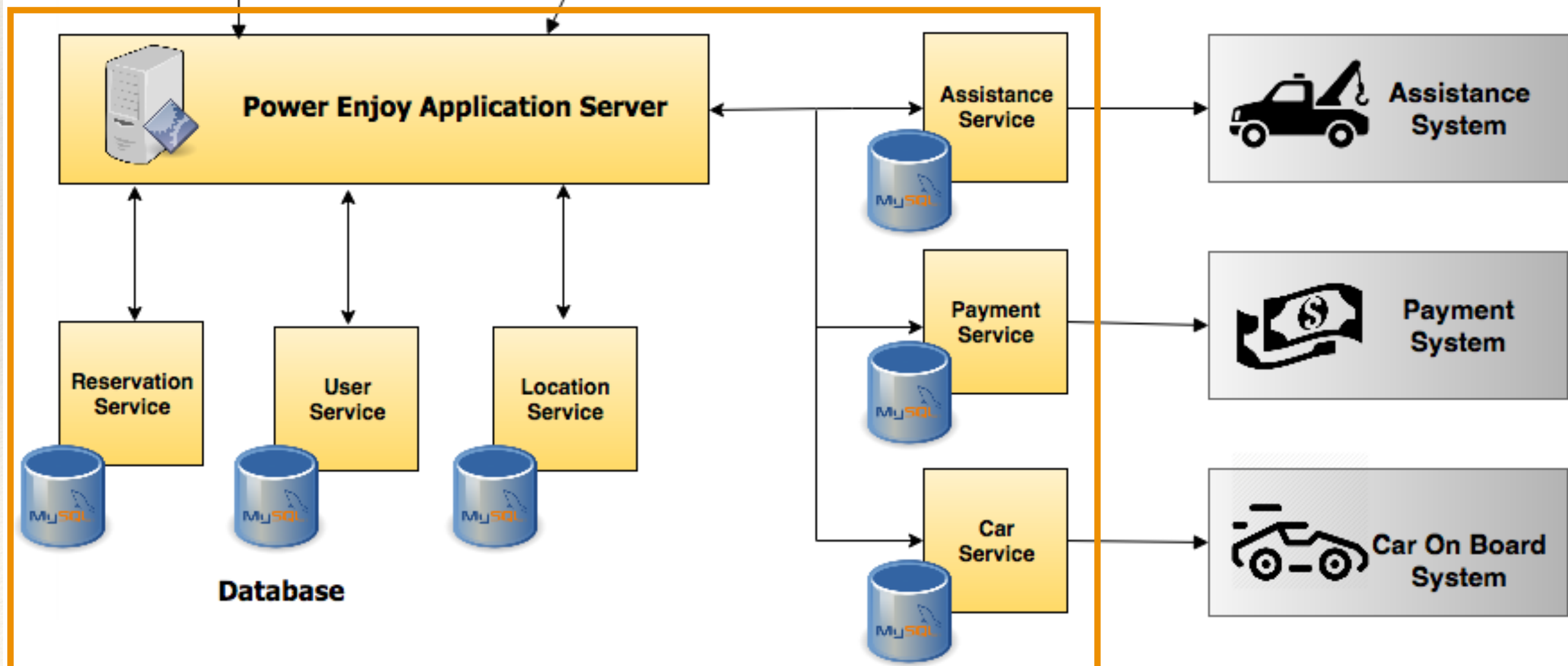
- ❖ It's a more **clean architecture**. Every component implements a service and provides an interface to all the other services.
- ❖ Changing / optimising each module **will not affect the whole system** as long as we maintain the same interface for each component.
- ❖ It's very flexible, it's will 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)

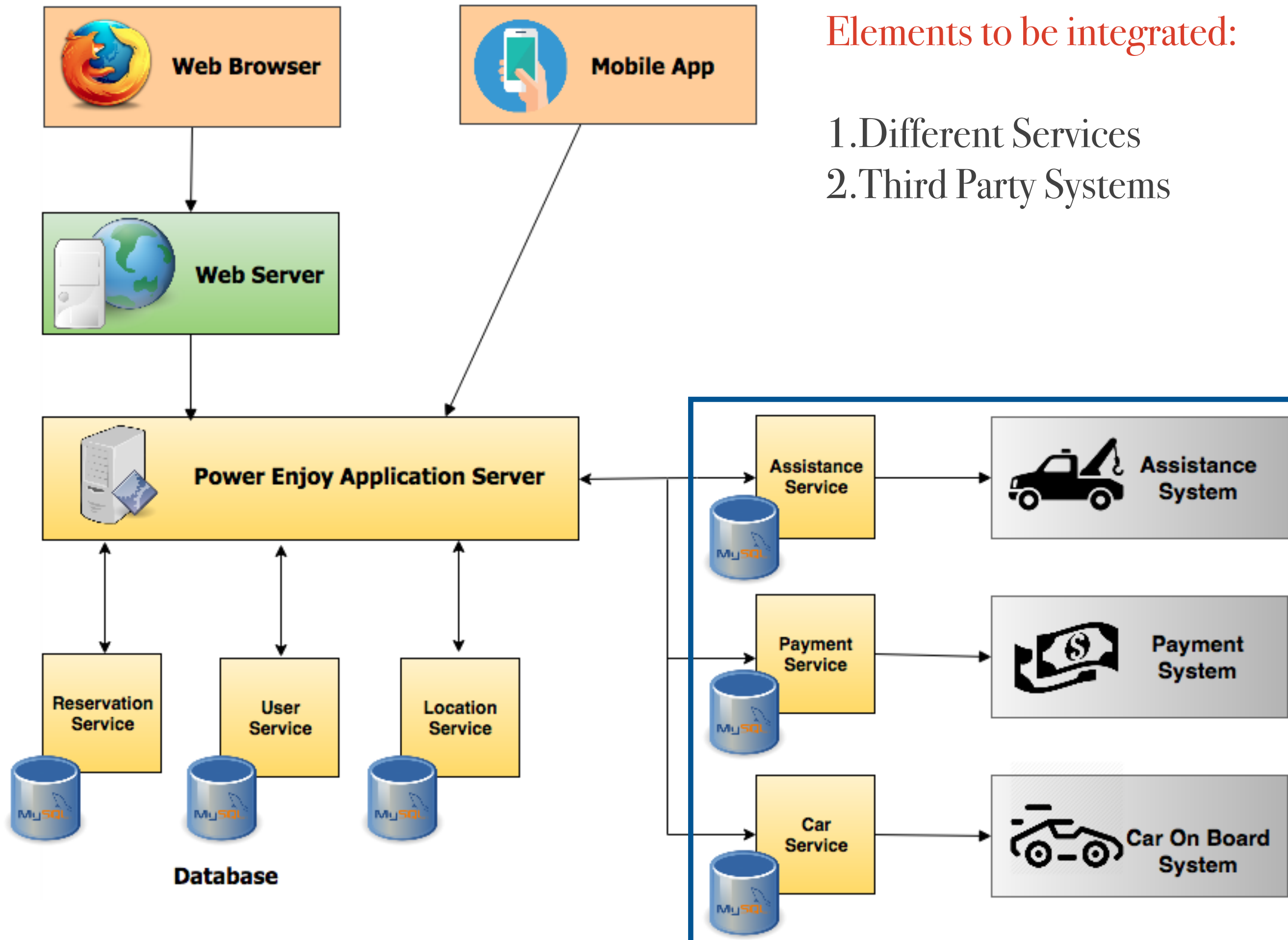


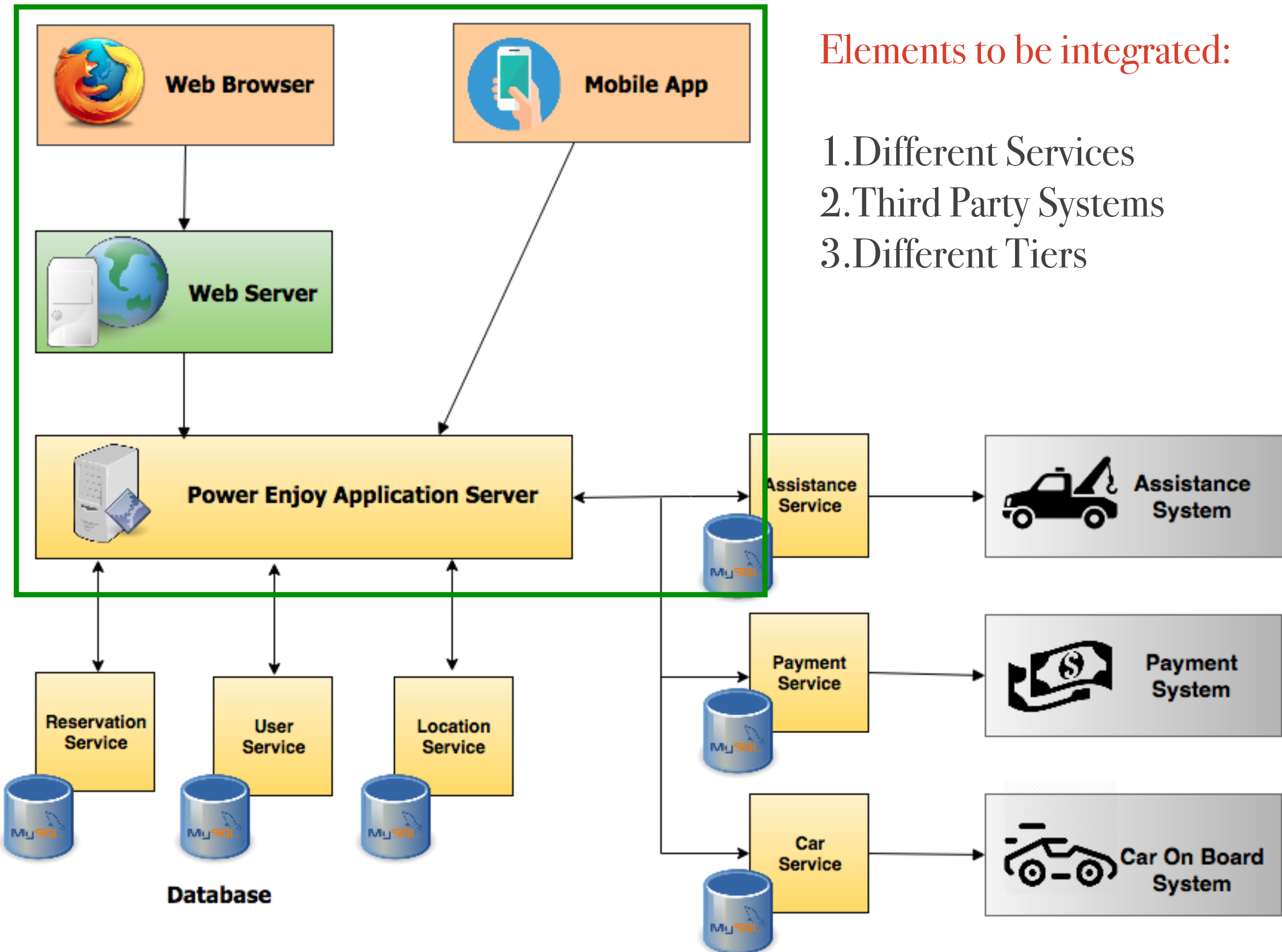


Elements to be integrated:

1. Different Services







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

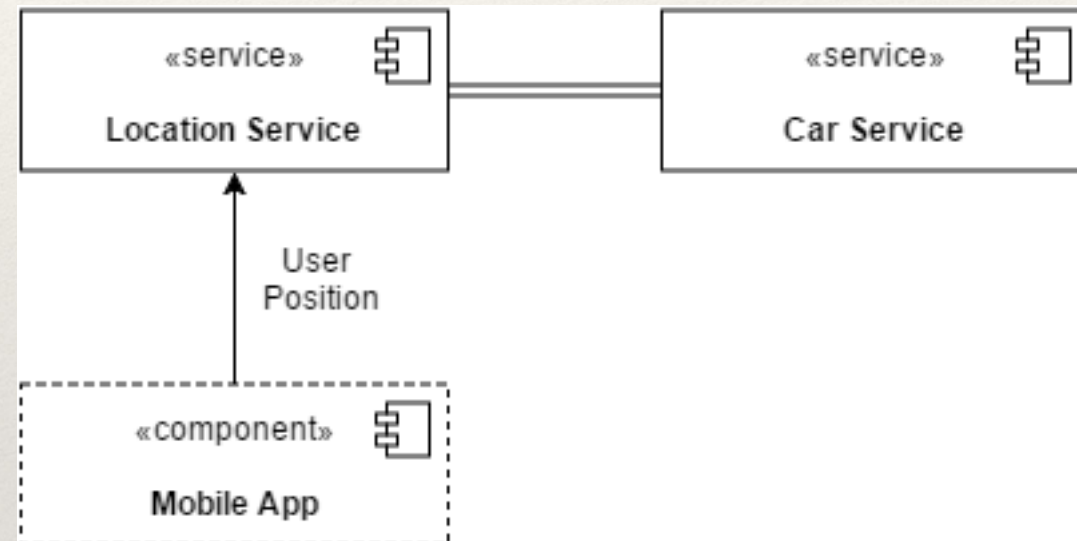
1. Ensure that services in relations with external systems works as expected
2. Ensure that we have control over the Car
3. Integration of Services
4. Integration with top layers
5. Alpha Test

Integration of Services

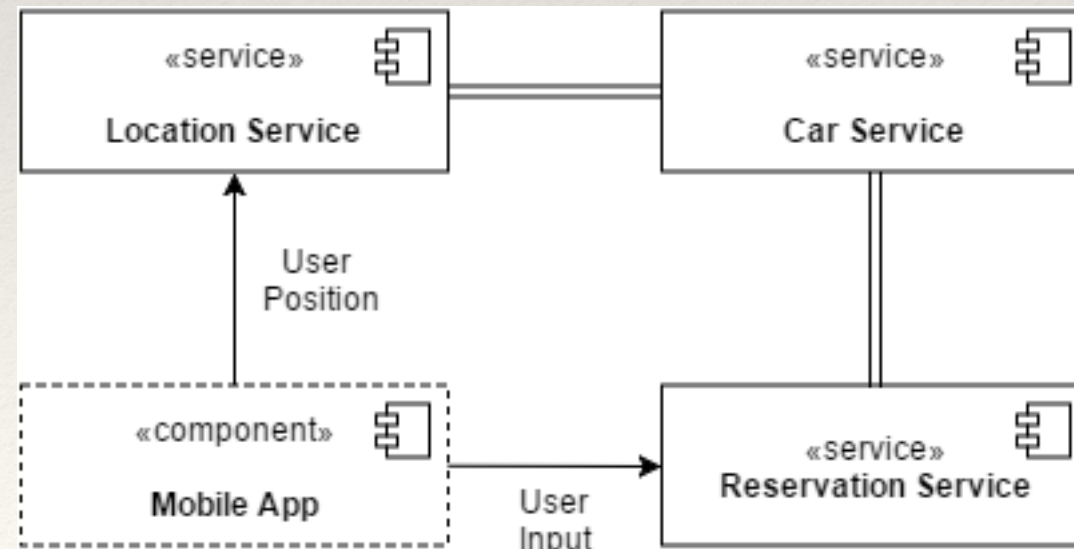
LOCATION OF VEHICLES



LOCATION OF USER

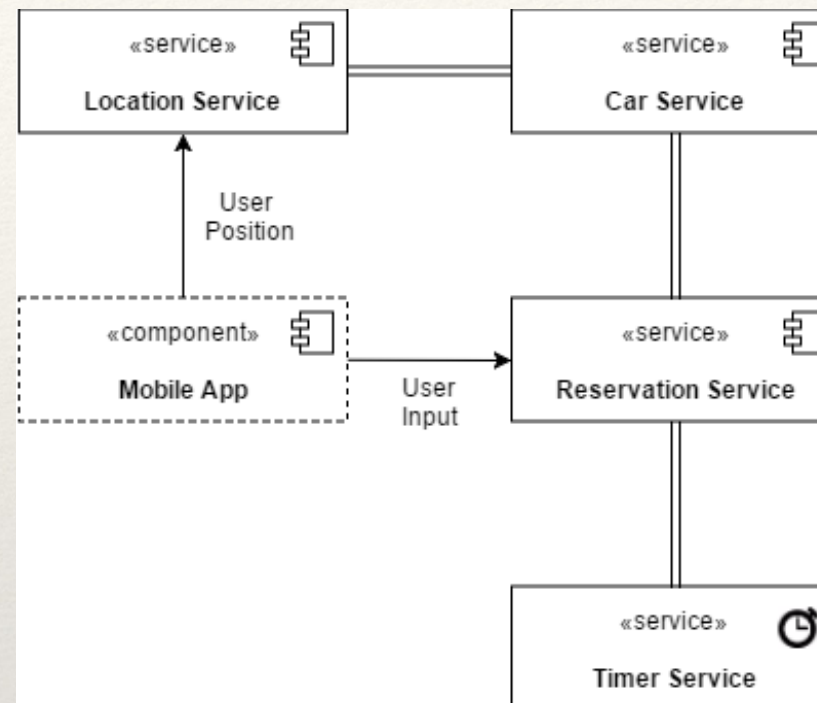


RESERVATION

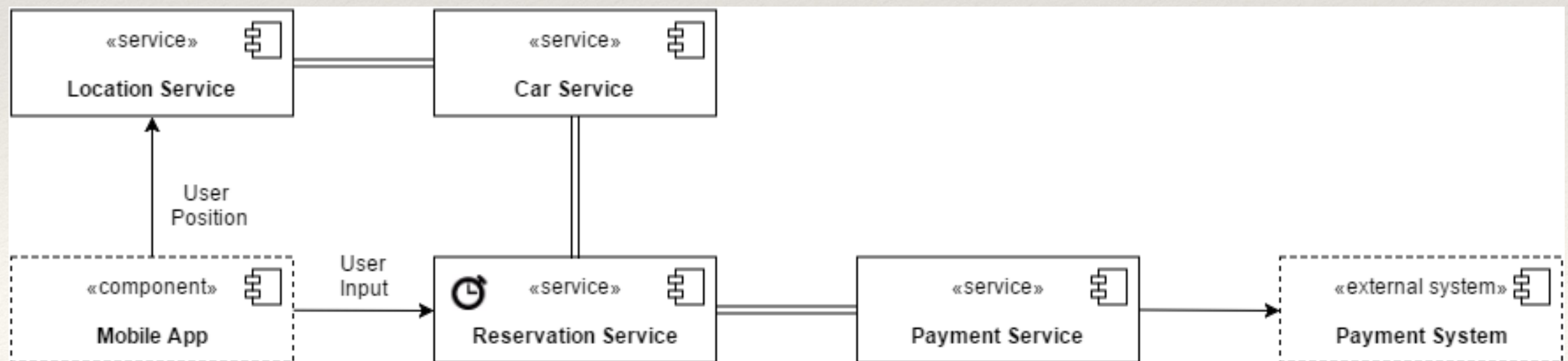


Integration of Services

TIMING

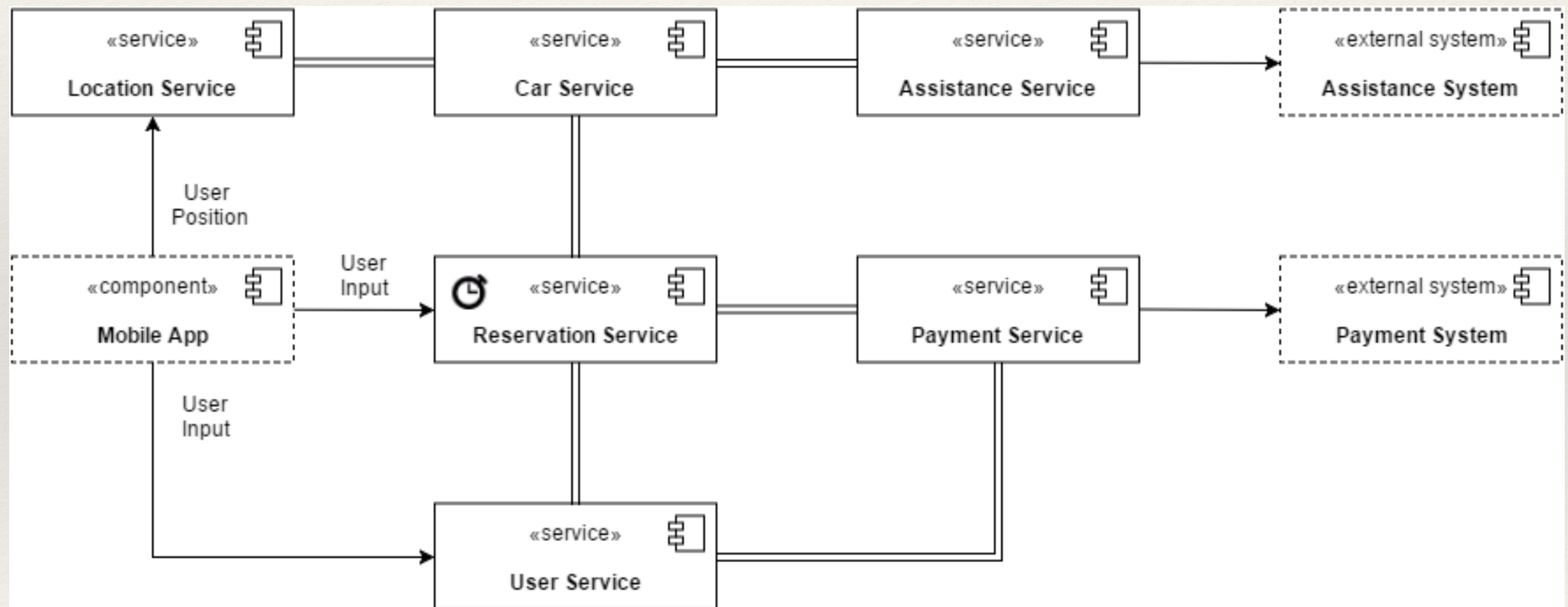


PAYMENT



Integration of Services

WHOLE SYSTEM



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 \approx 16 Months with 2 developers
 - ❖ High value but reasonable since Early Design approach

Scheduling

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

Project Risks

- ❖ Changing Requirements

- ❖ Why? Most recurrent and also unpredictable
- ❖ Strategy: Traceability Information and information hiding in design

- ❖ Personnel Shortfall

- ❖ Why? Just 2 developers
- ❖ Strategy: Positive Work Environment

Technical Risks

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

Business Risks

- ❖ Local regulations and policies
 - ❖ Regulations, regulations, regulations...
- ❖ Competitors
- ❖ Taxi Drivers Protests



Questions

