Software Engineering 2 Project
Design Document

PowerEnJoy

POLITECNICO
DI MILANO

Team: Nico Montali, Enrico Fini

Contents

Introduction

1.1 Purpose

The purpose of this document is to give a more detailed and technical overview about PowerEnJoy system.
This document is addressed to developers and aims to identify:

- The high level architecture;
- The design patterns;
- The main components and their interfaces provided one for another;
- The Runtime behaviour.

1.2 Scope

This document will present different level views in order to describe clearly the architecture of PowerEnJoy System. In particular we will present the component view, both high and low level, the runtime view and a further description of user interface, analysed in its runtime flow.
The System is based on two mobile applications and a web application.
As already explained in the RASD, the targets for the Service are:

- The Drivers;
- The Workers;
- The Admins.

The System allows the Drivers to reserve cars from both the mobile application and the web portal. At the same time it manages the the status of the cars and, if necessary, assigns tasks to the Workers via an ad-hoc mobile application. The Admins of the System can interact with the web portal to make privileged actions.

1.3 Definitions, Acronyms and Abbreviations

Driver Client of the system, i.e. the one that uses the service, reserves and drives. Every Driver must provide personal informations (name, surname, email, birthdate, a valid license and payment information;

Ride A single usage of the service, starts when the user turn on the engine, stops when the car is locked inside a Safe Area;

Blocked driver A driver that has an invalid license or invalid payment coordinates;

Worker Employee of the company, perform physical actions (moving, charging etc) on cars;

Report Car issue reported by the Driver during a Drive. Will later be assigned to a Worker;

Task Piece of work assigned to a worker. Different type of tasks are described later. Every task assigns a single car to a single worker;

Administrator Employee of PowerEni that is in charge of the administration of the system through the administration console;

Drop off The act of leaving the car inside a safe area. This ends the service provided to the driver and effectively make the driver pay;

Safe Area An area in which the Driver can park the car and stop the Service;

RASD Requirements Analysis and Specifications Document;

DD Design Document;

API Application Programming Interface;

DBMS DataBase Management System;

UI User Interface;

REST REpresantional State Transfer.

1.4 Reference Documents

- The RASD we released before;
- Sample Design Deliverable Discussed on Nov. 2.pdf.

1.5 Document Structure

Introduction: contains an overall description of the content and the structure of the document;

Architecture: this section contains:

- **Overview:** coarse view of the System i.e. the division in tiers;
- **High Level Components:** description of the main components of the application and their interaction;
- **Tracking Vehicle State:** more detailed description of the vehicle tracking mechanism;
- **Core Component:** describes the inner composition of the core component.

Algorithm Design: we will present some algorithms with pseudocode to show the logic of the most important parts of the System;

User Interface Design: starting from the ideas we gathered in the RASD we will give a deeper description of the UI;

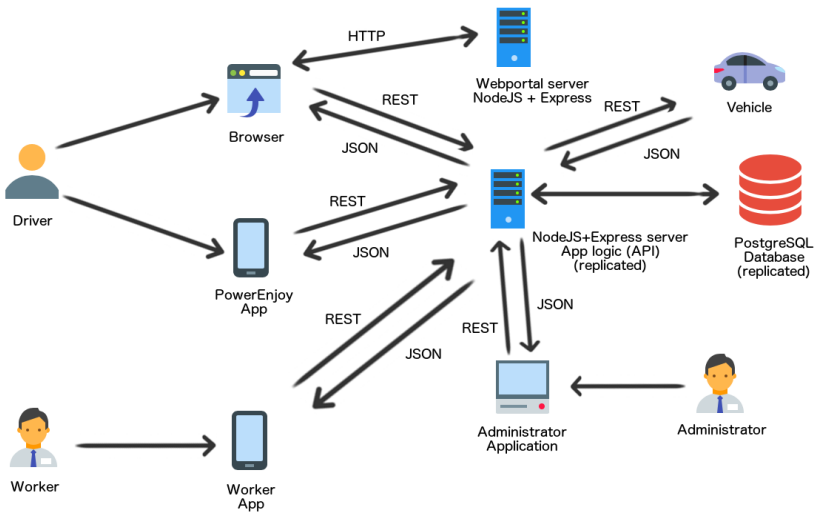
Requirements Traceability: An explanation of how our architectural choices respect the requirements described in the RASD.

Architecture

2.1 Overview

The PowerEnjoy system is designed as a 3-tier application:

- **Data tier:** A PostgreSQL DB, deployed on AWS and replicated for availability and integrity.
- **Application tier:** A NodeJS application that provides APIs, deployed on AWS and replicated. The Express library is used to realise API endpoints. These APIs are exposed in a REST manner, responses are JSONs. The replication of the application grants fault tolerance and, using a load balancer, also availability.
- **Presentation tier:** The presentation layer is realised with the 2 mobile applications (Driver and Worker), the web portal (Driver) and the Administrator application. The web portal is realised using EJS templating (server-side) and JQuery (client-side).

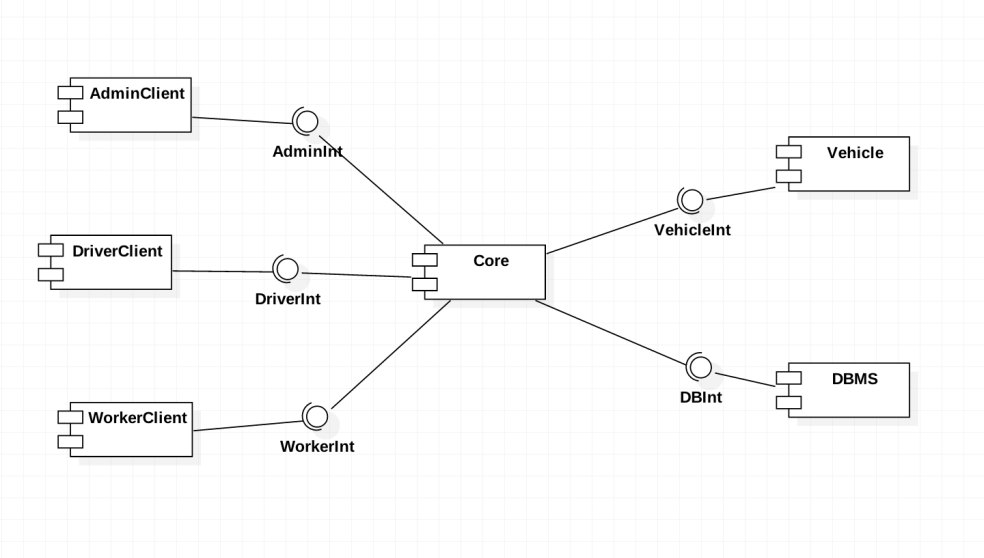


2.2 High level components

The high level structure of the System can be divided into 6 components. The main component is the **core**, that receives requests from **clients**, either **drivers**, **workers** and **administrators**; depending on the client type, the component provides different interfaces. The core also communicates with the **DBMS** component, responsible of data persistence. The core is also responsible of tracking vehicle states, by polling them every 30 seconds with a state request (HTTP GET). The core implements all the business logic:

- Process drivers requests and allocates the requested vehicle
- Manages vehicles, track their state
- Manages users (drivers signup, login)
- Manages reports and automatic tasks then it allocates them to workers

All the communications by clients (Driver, Worker, Admin) are realised in a synchronous client-server architecture, where the core is the server. The communications with the DBMS are also client-server (the core is the client) and are realised using PostgreSQL protocol. The communications with the vehicles uses HTTP but the core is the client, while vehicles are servers.



2.2.1 Tracking vehicle state

Tracking vehicle states requires a more detailed explanation. Our objective is to provide a simple interface for the core to communicate with vehicles

(both tracking and controlling), so that the core can perform simple GET requests to endpoints (e.g. /vehicle/vehicle-id) to know their state (e.g. position, occupants) or POST request to perform actions on vehicles (e.g. open/close doors). Every vehicle is equipped with a wireless communication system, and is connected to the internet (so it has a public and static IP address). The core associate every vehicle ID with its IP address (and store this address in the DB), so every time a request to a specific vehicle is made, the core resolves the IP address of the requested vehicle. For security reasons, vehicles accept request only from the core IP (that is static). Every vehicle implements a simple on-board server, that answers to the core GET/POST requests, i.e. providing the vehicle's state or performing actions. The vehicle can also act as a client, sending requests to specific endpoints on the core to notify either the end of a drive or some detected issue.

2.3 Core component

Since the other components are quite simple and atomic, only the core component need some further analysis. The core has many functionalities that are divided in different components (for component diagram, see picture XXX):

- **Vehicle Manager** This component is responsible of tracking vehicle usage state (see state chart, picture XXX) but also vehicle internal state (e.g. position). This states are saved to the DB to provide persistency but also analytics. It provides an interface to the Drive Controller and the Task Manager to get information about the car and to update their state. In order to be updated about the state of the vehicles the Vehicle Manager keeps polling them.

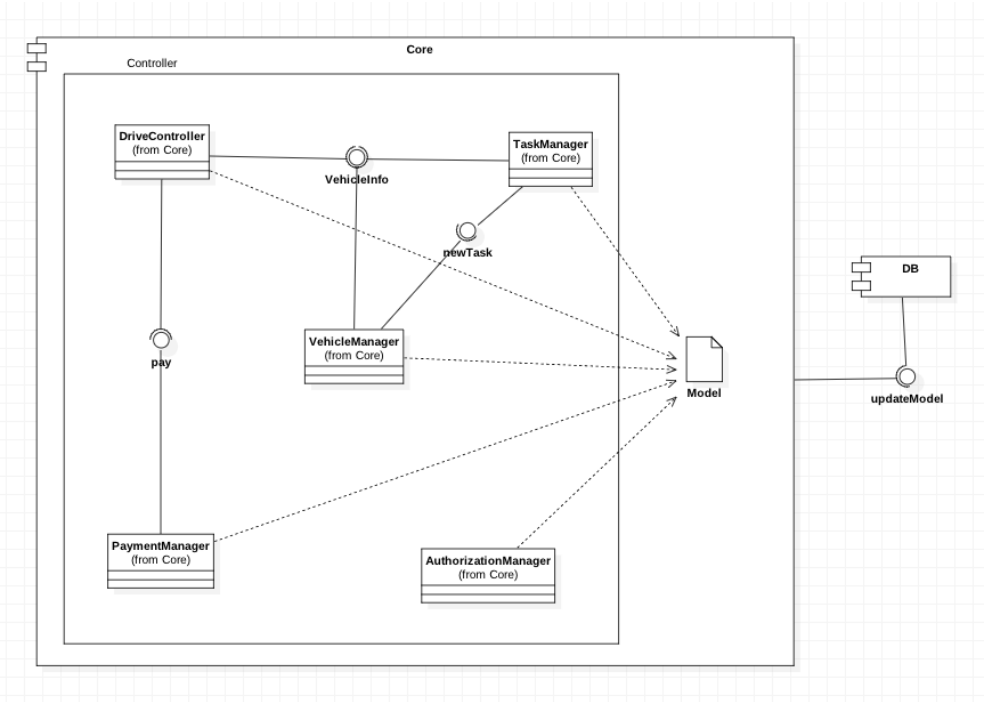
- **Drive Controller** This component is responsible of processing Drivers requests (e.g. reservation, cancellation, termination). It communicates with the Vehicle Manager to update vehicle state. It also communicates with the Payment Manager to process Drivers payments. It is also responsible of computing the correct amount due at the end of a drive. Once the drive has ended and the data is sent to server by the vehicle it computea the base amount to pay and applies discounts. A pseudo-code implementation of the amount computing is provided in chapter XXX.

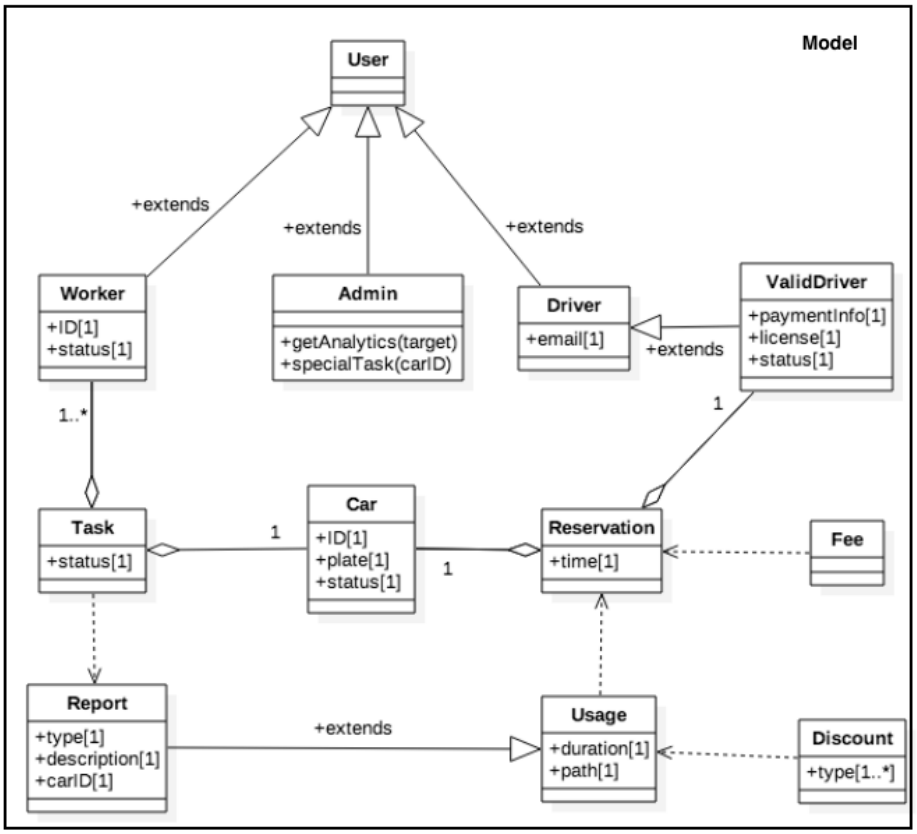
- **Payment Manager** It receives request to make a specific user pay a specific amount, process the payment and store it in the DB if the

payment was successful. It is also responsible of verifying with the pre-authorization if the Driver can pay.

- **Task Manager** It is primarily responsible of managing task allocation to workers, but also of processing Drivers reports and Admins special tasks queries. It exposes an interface to the Vehicle Manager too in order to allow it to perform automatic maintenance tasks on vehicles. A pseudo-code implementation of the worker choosing algorithm is provided in chapter XXX.

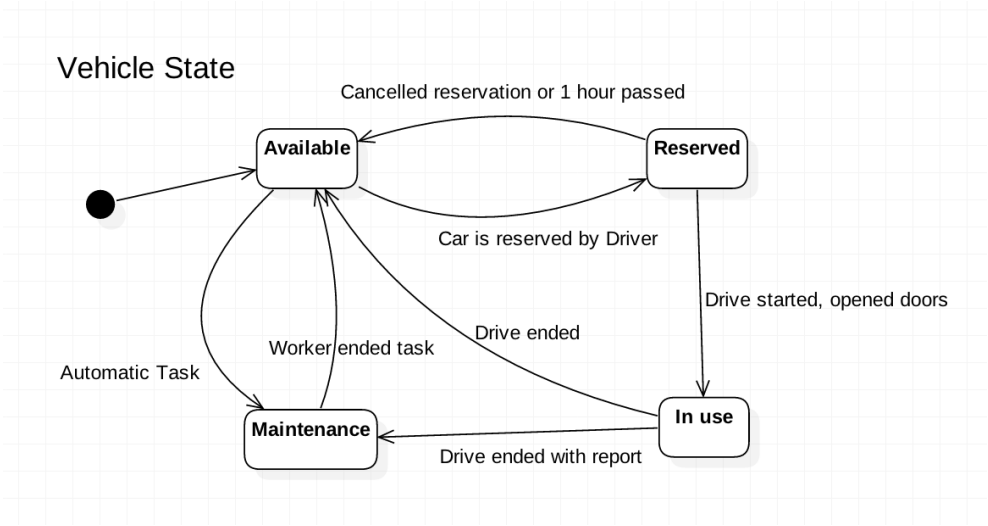
- **Authorization Manager** It manages most of the security of the system. It is responsible of Drivers and Workers signup and login. It is also responsible of verifying that every requests to the core are performed by an authorized user (e.g. Driver can't request Workers API).

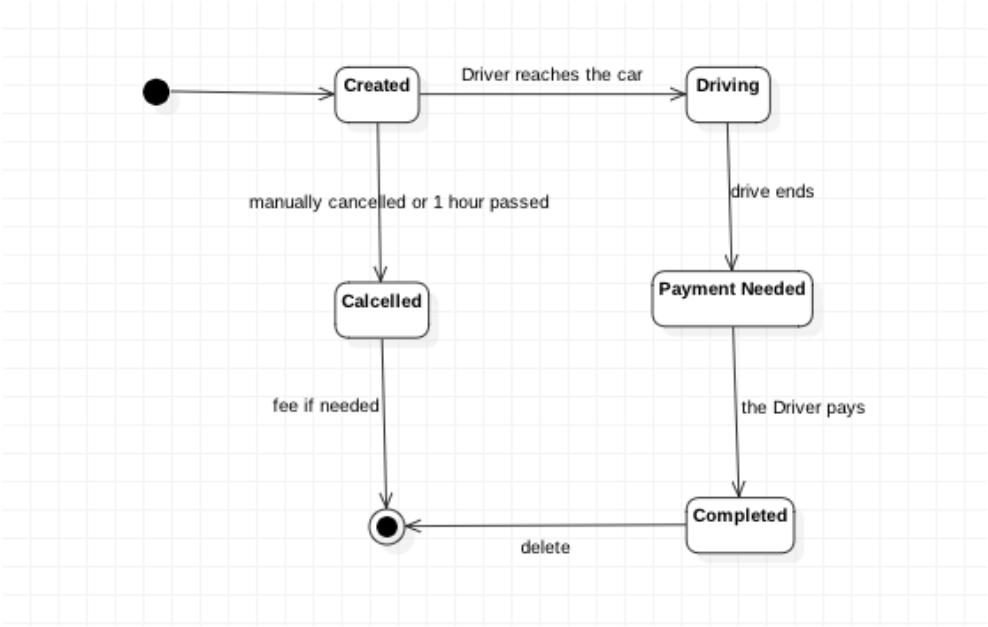


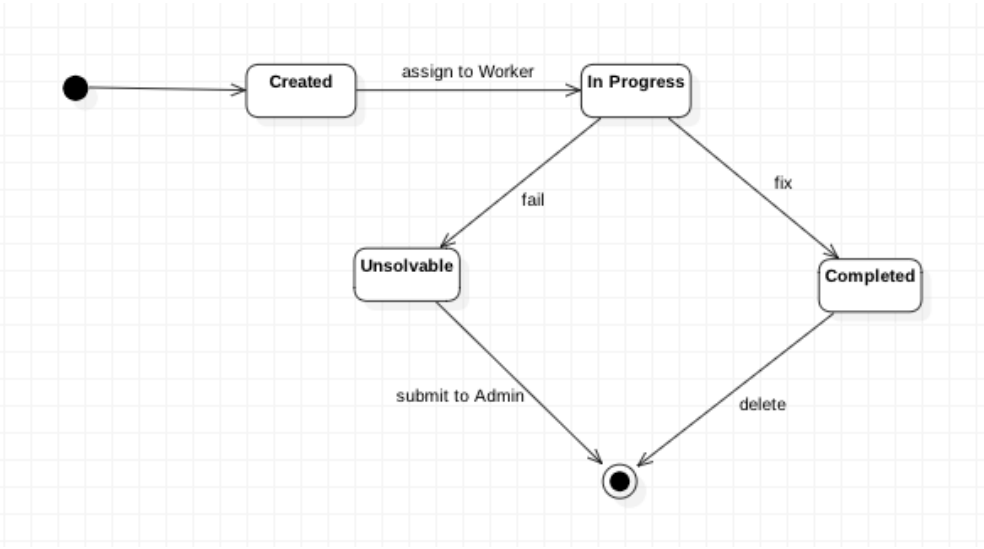


2.3.1 Relevant State Charts

To give a deeper explaination of the dynamic behaviour of the model we add here some State Chart Diagrams, describing the lifecycle of vehicles, reservations and tasks:


Vehicle State
Cancelled reservation or 1 hour passed
Available
Reserved
Car is reserved by Driver
Drive started, opened doors
Drive ended
Worker ended task
Automatic Task
Maintenance
In use
Drive ended with report


Created
Driver reaches the car
Driving
manually cancelled or 1 hour passed
drive ends
Calcelled
Payment Needed
fee if needed
the Driver pays
Completed
delete


Created
assign to Worker
In Progress
fail
fix
Unsolvable
Completed
submit to Admin
delete