



POLITECNICO MILANO 1863

Politecnico di Milano
A.A. 2016–2017
Software Engineering 2: “PowerEnJoy”
Design Document

Pietro Ferretti, Nicole Gervasoni, Danilo Labanca

December 3, 2016

Contents

1	Introduction	3
1.1	Purpose	3
1.2	Scope	3
1.3	Definitions, Acronyms, Abbreviations	4
1.3.1	Definitions	4
1.3.2	Acronyms	5
1.3.3	Abbreviations	5
1.4	Reference Documents	5
1.5	Document Structure	6
2	Architectural Design	7
2.1	Overview	7
2.2	Component View	7
2.3	Deployment View	9
2.4	Runtime View	12
2.5	Component Interfaces	12
2.6	Selected Architectural Styles and Patterns	13
2.6.1	Protocols	13
2.6.2	Patterns	13
2.7	Other Design Decisions	13
3	Algorithm Design	14
4	User Interface Design	16
4.1	Mockups	16
4.2	UX Diagrams	16
4.3	BCE Diagrams	16
5	Requirements Traceability	17
6	Effort Spent	18
7	Revisions	19
7.1	Changelog	19

1 Introduction

1.1 Purpose

The purpose of this document is to specify in a more technical way how our system is designed to be built and work. This document is intended for developers, designers and all interested stakeholders as a guideline for the implementation of the system.

This document will first of all illustrate the system architecture, the components that make it up and how each of its components is designed to interact with each other.

Furthermore this document will describe the behaviour of the system and the applications, from both the back-end and the front-end side.

Finally this document will describe the patterns we used to design the system and how each of them comes into play.

1.2 Scope

The aim of this project is to specify in detail a new digital management software for PowerEnJoy, a car-sharing service that employs electric cars only.

PowerEnjoy will offer a very valuable service to its users, letting them borrow cars to drive around the city freely, as an alternative to their own vehicles and public transport. Among the advantages of using PowerEnJoy we can note being able to find available cars in any place that is served by our system and having dedicated spots to park in (namely, PowerEnJoy's power grid stations). Furthermore, thanks to the fact that all the cars that we provide are electrically powered, PowerEnJoy is also very environmentally friendly.

PowerEnJoy's users, after registering, will be able to reserve, unlock and drive the cars our system will provide. Users will be charged per minute until they park the car in a safe area and end the ride. Users will be able to park their car temporarily and use it again later, or end their ride remotely.

Our system will incentivize virtuous behaviour by offering several discounts if certain conditions are met (like charging a car at a power grid station).

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Definitions

- *Guest*: a person that is not registered to the system.
- *User*: a person that is registered to the system. Users can log in to the system with their email or username and their password. Their first name, last name, date of birth, driving license ID are stored in the database.
- *Safe area*: a location where the user can park and leave the car. Users can end their ride and park temporarily only in these locations. The set of safe areas is predefined by the system.
- *Power grid station*: a place where cars can be parked and plugged in. While a car is plugged in a power grid station its battery will be recharged. Power grid stations are by definition safe areas.
- *Available car*: a car that is currently not being used by any user, and has not been reserved either. Available cars are in good conditions (not dirty nor damaged) and don't have dead batteries.
- *Reservation*:
 - the operation of making a car reserved for a user, i.e. giving permission to unlock and use the car only for that user, forbidding reservations by other users.
 - the time period between the moment a reservation is requested and the moment the user unlocks the car, or the reservation is canceled.
- *Ride*: the time period from the moment a reserved car is unlocked to the moment the user notifies that he wants to stop using the car and closes all the doors. A ride doesn't stop when a car is temporarily parked, but continues until the user chooses to leave the car definitely.
- *Possession*: users that have reserved and unlocked a car are said to have possession of the car. While a user has possession of a car they are the only person that can drive it, lock or unlock it, and no other person can take possession of it until the user frees it. Users lose possession of a car when their ride ends.
- *Temporary parking*: the act of parking a car in a safe area and, after notifying the system, locking it and leaving it for a finite amount of time. The user that does this retains the right to use the car and can unlock it later to use it again.
- *Bill*: a record of the money owed by the user at the end of a ride.
- *Outstanding bill*: a bill that hasn't been paid yet.

- *Suspended user*: a user that cannot reserve or use cars. Usually users are suspended because they have outstanding bills.
- *Payment method*: a way to transfer money from the user to the system. Our system will only accept credit cards and online accounts like Paypal.
- *Payment API*: an interface to carry out money transactions, offered by the external provider associated to the payment method used (e.g. a bank).
- *CAN bus*: a vehicle bus standard designed to allow micro controllers and devices to communicate with each other.

1.3.2 Acronyms

- **DD**: Design Document
- **RASD**: Requirements Analysis and Specification Document
- **DB**: Database
- **CVV**: Card Verification Value
- **DOB**: Date of birth
- **PGS**: Power Grid Station
- **GPS**: Global Positioning System
- **CAN bus**: Controller Area Network bus

1.3.3 Abbreviations

- **[Gx]**: Goal
- **[RE.x]**: Functional Requirement
- **[UC.x]**: Use Case

1.4 Reference Documents

- IEEE Std. 1016-2009, “IEEE Standard for Information Technology – Systems Design – Software Design Descriptions”
- ISO/IEC/IEEE Std. 42010:2011, “Systems and software engineering – Architecture Description”
- Specification document: “Assignments AA 2016-2017.pdf”

1.5 Document Structure

This document is structured as follows:

- **Section 1 – Introduction:** this section introduces the design document. It contains a justification of his utility and indications on which parts are covered in this document that are not covered by the RASD.
- **Section 2 – Architectural Design:** this section is divided into more parts:
 - **Overview:** this section shows a high-level view of our system’s architecture.
 - **Component View:** this section gives a more detailed (in-depth) view of the components of the application and how they communicate.
 - **Deployment View:** this section shows the way the components will be deployed to run our application.
 - **Runtime View:** in this section we use sequence diagram to show the flow of the most important functionalities of the system.
 - **Component Interfaces:** the interfaces between the components are presented in this section.
 - **Selected Architectural Styles and Patterns:** this section explains the architectural choices taken during the creation of the application.
 - **Other Design Decisions**
- **Section 3 – Algorithm Design:** this section describes the most critical parts via some algorithms. Pseudo code is used in order to hide unnecessary implementation details in order to focus on the most important parts.
- **Section 4 – User Interface Design:** this section presents mockups and user experience explained via UX and BCE diagrams.
- **Section 5 – Requirements Traceability:** this section aims to explain how the requirements identified in the RASD are linked to design elements.
- **Effort Spent**
- **References**
- **Revisions**

2 Architectural Design

2.1 Overview

2.2 Component View

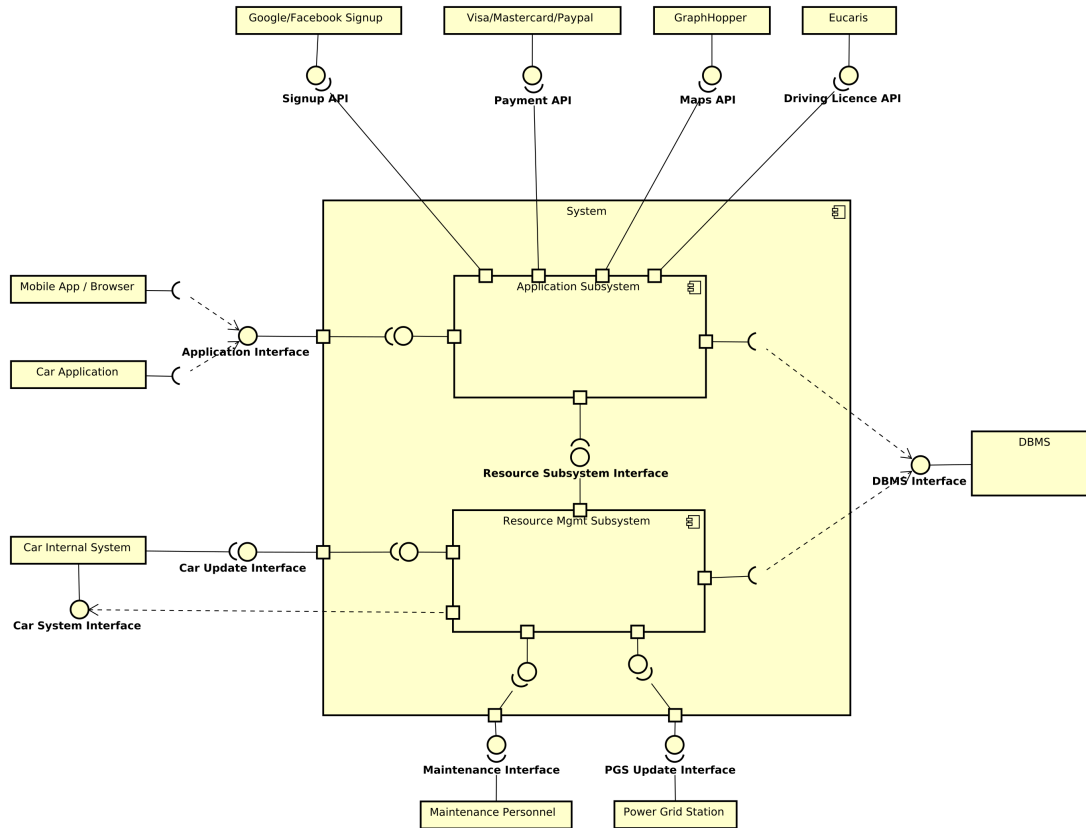


Figure 1: Component Diagram 1

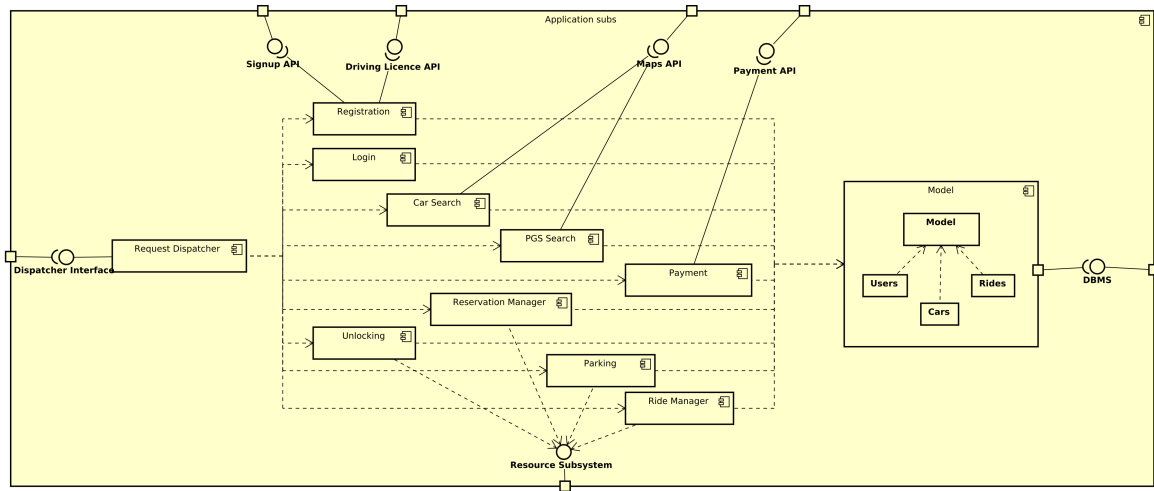


Figure 2: Component Diagram 2

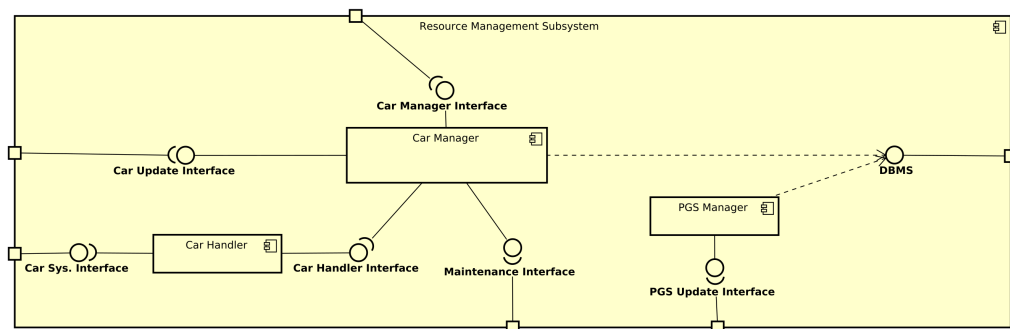


Figure 3: Component Diagram 3

2.3 Deployment View

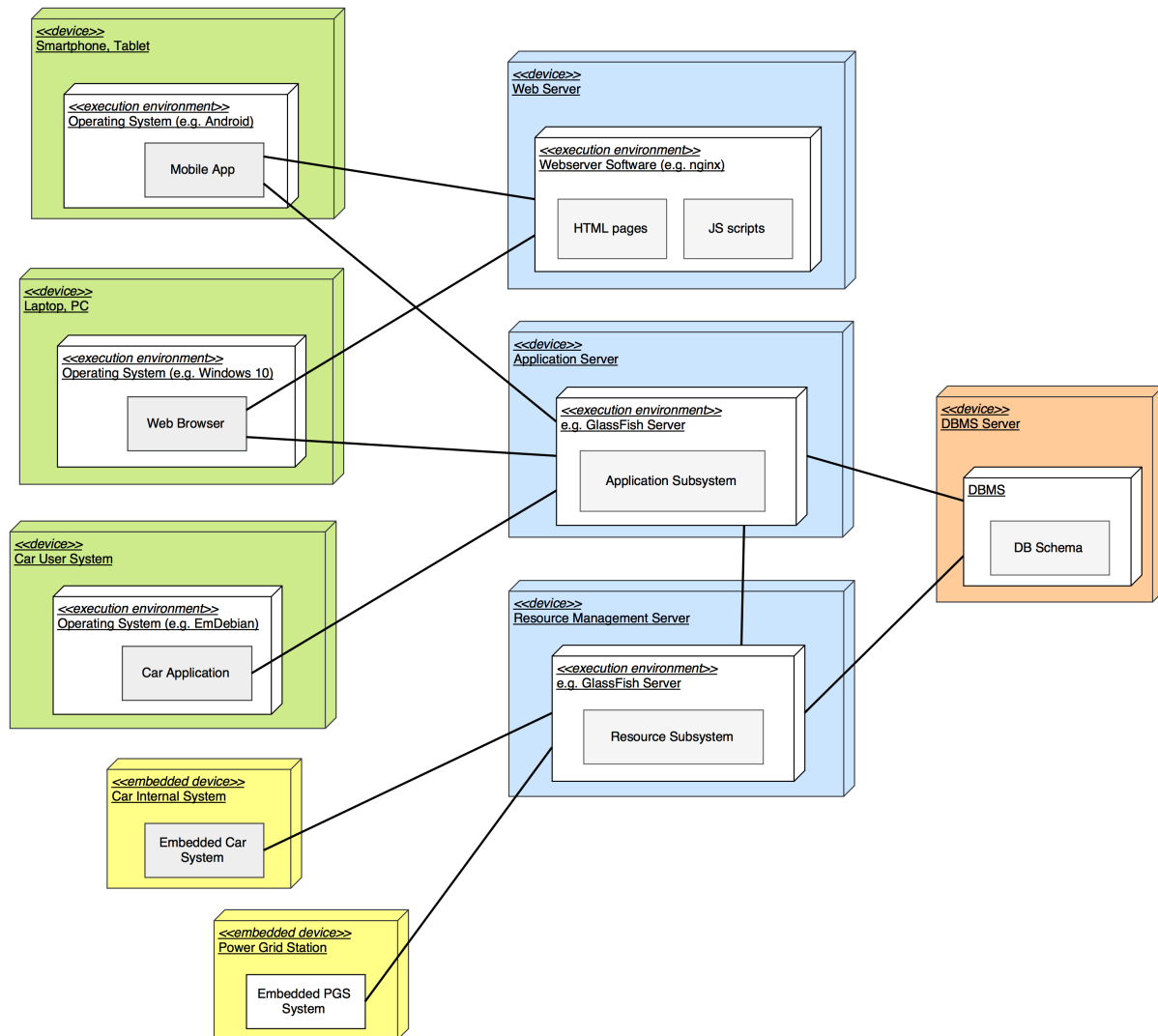


Figure 4: Deployment Diagram

tre tier: client, business logic, database

client: - app e browser si collegano al web server per avere le pagine web, poi si collegano all'applicazione per raccogliere i dati - l'applicativo sulla macchina non è basato sul web e si collega direttamente all'applicazione server per avere tutto - esiste un "resource management server" che raccoglie informazioni (gps, ecc.) dalle macchine e disponibilità posti dalle PGS - l'applicazione server inoltre può inviare comandi alle macchine tramite il resource server the car is

listening for commands - application server e resource server sono collegati al DBMS che contiene tutto in tempo reale

2.4 Runtime View

2.5 Component Interfaces

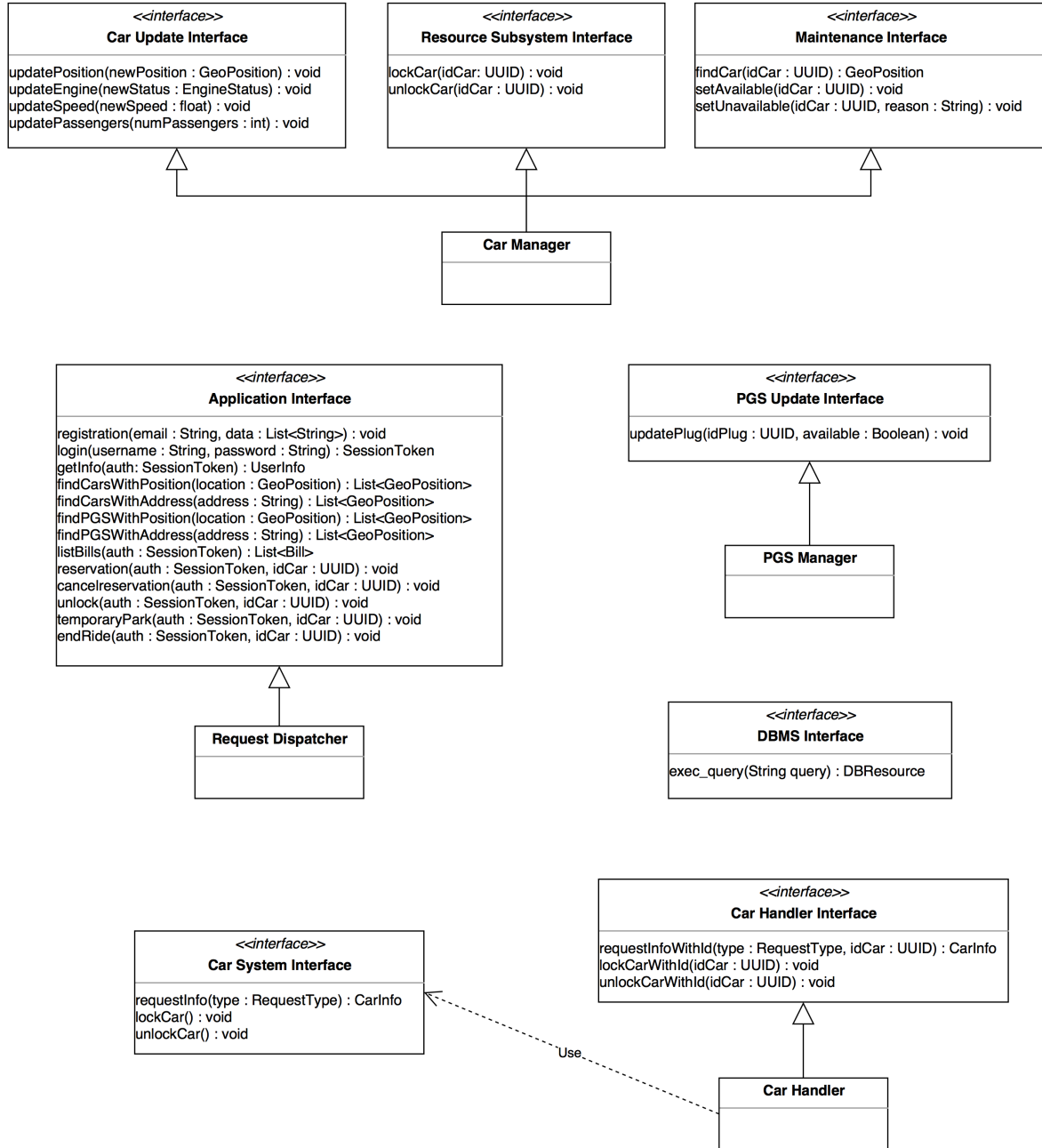


Figure 5: Component Interfaces

2.6 Selected Architectural Styles and Patterns

2.6.1 Protocols

2.6.2 Patterns

2.7 Other Design Decisions

3 Algorithm Design

```
1  package powerEnJoy;
2
3  import java.util.List;
4
5  import graphHopper.GraphHopper;
6
7  public class Discounts
8  {
9      private List<PowerGridStation> pgsList;
10     private final static int MAXDISTANCE = 3000;
11
12     public int calculatesDiscount(int amount, Car car, Ride ride)
13     {
14         int minDistance = 1000000000;
15         int distance = 0;
16         for(PowerGridStation pgs : pgsList)
17         {
18             distance = GraphHopper.distance(pgs.getPosition(), car.getPosition());
19             if(distance < minDistance)
20                 minDistance = distance;
21         }
22
23         int discountPercentage = 0;
24
25         if(car.getPassengers() > 1)
26             discountPercentage += 10;
27         if(car.getBattery() > 50)
28             discountPercentage += 20;
29         if(car.isCharging())
30             discountPercentage += 30;
31         if(car.getBattery() < 20 || minDistance > MAXDISTANCE)
32             discountPercentage -= 30;
33         if(ride.isMso() && (ride.getChosenPGS().getPosition().equals(car.getPosition()
34             ())))
35             discountPercentage += 10;
36
37         int discount = amount * (discountPercentage/100);
38
39         return (amount - discount);
40     }
41 }
```

```

1  package powerEnJoy;
2
3  import graphHopper.*;
4
5  import java.util.ArrayList;
6  import java.util.HashMap;
7  import java.util.List;
8  import java.util.Map;
9  import java.util.Map.Entry;
10
11 public class MoneySavingOption
12 {
13     private final static int TIMEDISTANCE = 300;    //300 seconds = 5 minutes
14     private final static int SPACEDISTANCE = 500;    //500 metres
15
16     private List<PowerGridStation> pgsList;
17     private List<Car> availableCar;
18
19     public PowerGridStation moneySavingOption(String address)
20     {
21         Map<PowerGridStation, Integer> count = new HashMap<>();
22         GeoPosition destination = GraphHopper.geocoding(address);
23         List<PowerGridStation> accetablepgs = new ArrayList<>();
24
25         //looks for the power grid stations reachable in a reasonable time
26         for(PowerGridStation pgs : pgsList)
27             if(GraphHopper.isochrone(destination, pgs.getPosition()) < TIMEDISTANCE)
28                 accetablepgs.add(pgs);
29
30         //estimates the "density" of the cars near a power grid station
31         for(PowerGridStation pgs : accetablepgs)
32         {
33             int i = 0;
34
35             for(Car car : availableCar)
36                 if(GraphHopper.distance(car.getPosition(), pgs.getPosition()) <
37                     SPACEDISTANCE)
38                     i++;
39
40             count.put(pgs, i);
41         }
42
43         PowerGridStation chosenPGS = null;
44         int min = 1000000000;
45
46         //finds the power grid station with the minimum "density"
47         for(Entry<PowerGridStation, Integer> entry : count.entrySet())
48             if(entry.getValue() < min)
49                 chosenPGS = entry.getKey();
50
51         return chosenPGS;
52     }

```

4 User Interface Design

4.1 Mockups

4.2 UX Diagrams

4.3 BCE Diagrams

5 Requirements Traceability

6 Effort Spent

7 Revisions

7.1 Changelog

-