CampDays

System Design

Version 1.0

13.04.2018

Orhan Aykut Kardeş

Muhammed Ali Altınel

Hüseyin Haluk Nurbaki

Berke Kağan Yıldız

Batuhan Kurt

Mert Günay

Prepared for

SE302 Software Engineering



Table of Contents

[1. Introduction 1](#_Toc433996772)

[1.1. Purpose of the System 1](#_Toc433996773)

[1.2. Design Goals 2](#_Toc433996774)

[1.3. Definitions, Acronyms, and Abbreviations 3](#_Toc433996775)

[1.4. References 3](#_Toc433996776)

[2. Current Software Architecture 4](#_Toc433996777)

[3. Proposed Software Architecture 4](#_Toc433996778)

[3.1. Overview 4](#_Toc433996779)

[3.2. System Decomposition 5-6](#_Toc433996780)

[3.3. Hardware Software Mapping 7](#_Toc433996781)

[3.4. Persistent Data Management 8](#_Toc433996782)

[3.5. Access Control and Security 9-10](#_Toc433996783)

[3.6. Global Software Control 10](#_Toc433996784)

[3.7. Boundary Conditions 11](#_Toc433996785)

[4. References 12](#_Toc433996787)

SYSTEM DESIGN DOCUMENT

# Introduction

Our Campdays system design is quite simple but very functional. This web-site provides to help the people who want to find real camping places. This website includes several quality camping places. Visitors can easily find what they want with our functional system design. After the design stage, we can decrease the time that required the implementation.

## Purpose of the System

As we said in RAD, this project is a camping place reservation web-site where users can find the camp places according to their demands. This system is unique for our country, people who want to camp can easily reach the camping places. The main purpose of this project helping to people for finding real camping places. This website includes many quality camping places. Camp owners can easily add their camp page to the system with admin approval. This system also supplies that every user can make advance search. Advance search ensures finding places that near to user who uses it. In addition, users can make a reservation from a camping place with using the pinned map which is on main page. As a result, this system is unique and, useful to find and to make reservation a camping place.

## Design Goals

The purpose of our project is to fulfill the requirements of software engineering completely. This means that we specify all functional and nonfunctional functions together. With the definition of the functions, we have defined all the requirements for our Campdays project and have prepared an infrastructure for future versions or new projects. In our Campdays system design, we provide to our users or our visitors with easy access to our website to search for real camping places with camp names, price, region, and camping type. In addition, users can use blog for commenting a camp place, also they can be friend with sending follow request to each other. Search near feature uses for searching camp places that near to users, it searches by location. The features our system evaluates based on non-functional functions are as follows:

* **Dependability**

One of the most significant non-functional requirement is system security. The user security is on the front board in our Campdays system. In addition to security and safety, we paid attention to robustness, reliability, availability and fault tolerance criteria to make a complex system.

* **Maintenance**

Maintenance of your Campdays system is periodically performed by the administrator. Of course, when we are creating to our system, shortcomings such as extensibility, modifiability, adaptability, portability and readability were taken into consideration.

* **End User Criteria**

On our Campdays website, users and visitors can search, sign up, advance search, search near, view their own information, and make reservation for a camping place. Our system efficiently stores and retrieves user data in a dynamic manner. In addition, we noticed that utility and usability factors are important for us. The Campdays system supports Microsoft and MacOS operating systems.

* **Performance**

Our Campdays system is responsive and it can accomplish a maximum number of tasks easily. The memory space of our system is available for speed optimizations. As we mentioned the response time, through put and memory criteria are significant for our system.

* **Cost**

We try to accomplish optimal level for cost of our system when we develop it. Also, this cost not only for design considerations but managerial, as well. The maintaining backward compatibility with a previous system can add to the development cost while reducing the transition cost. By the way, we handle the development cost, deployment cost, upgrade costs, maintenance cost and administration cost.

## Definitions, Acronyms, and Abbreviations

The abbreviations and definitions contained in the document are given below:

* Campdays: Location based camping area finder
* Admin: System Admin
* Owner: Owner of a camping place on the site
* User: Uses the site
* Model: A schematic description of a system that accounts for its known or inferred properties.
* System: Any interacts by the application are considered to be done by the system.
* Efficiency: The properties of an algorithm, which is the amount of computational resources used by the algorithm.
* Service: Service is a keyword. Purpose of the service is to provide the customer with a secure payment system.
* OOP: Object Oriented Programming
* POP: Procedural or Produce Oriented Programming
* SQLite: Structured Query Language Lite
* HTML: Hypertext Markup Language
* CSS: Cascading Style Sheets
* MIT License: Massachusetts Institute of Technology License
* API (Django): Application Programming Interface
* SDD: System Design Document.

## References

Requirements Analysis Document (16.03.2018)

<https://www.djangoproject.com>

<https://developer.mozilla.org/en-US/.../Django/Introduction>

<https://github.com/django>

# Current Software Architecture

Our Campdays system is a smart system and, we designed our system on the web. Our system differs from other sites, it has owner user type that allows camping place owners to display their places in the site, and also people can make a reservation.

The main idea is finding camping places throughout map and gps system. The communication between the system administrators, owners and users works in a synchronous manner. The platform is running synchronously, the host and system administrator are very quick to inform. As a result, we used the MVC (Model, View, Controlled) architecture style for our Campdays system. Because, MVC is well suited for interactive systems, especially when multiple views of the same model are needed.

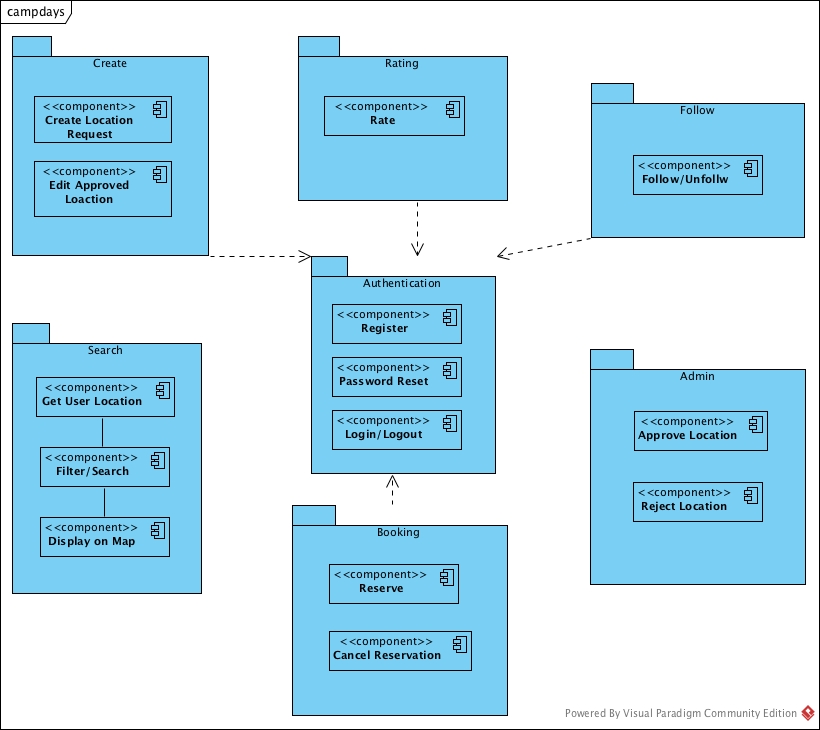
# Proposed Software Architecture

Documents the requirements elicitation and the analysis model of the new system. The Campdays system is web based. Our project will be very useful for people who are looking to find and compare camping placeses near to them or any location that they desire. Our system has different features than other similar systems. For example, other systems works with hotel, hostels and rental apparments but our system works with camping places that is not really easy to find. Our Campdays system also serves not only for members, but also for users who visit our system without a membership. They can not make a reservation but they can use the map feature to find places near them. In addition, our system contains all the features that exist in similar systems, for example advance search, easy reservation, commenting to places, simple membership and a clear interface.

## Overview

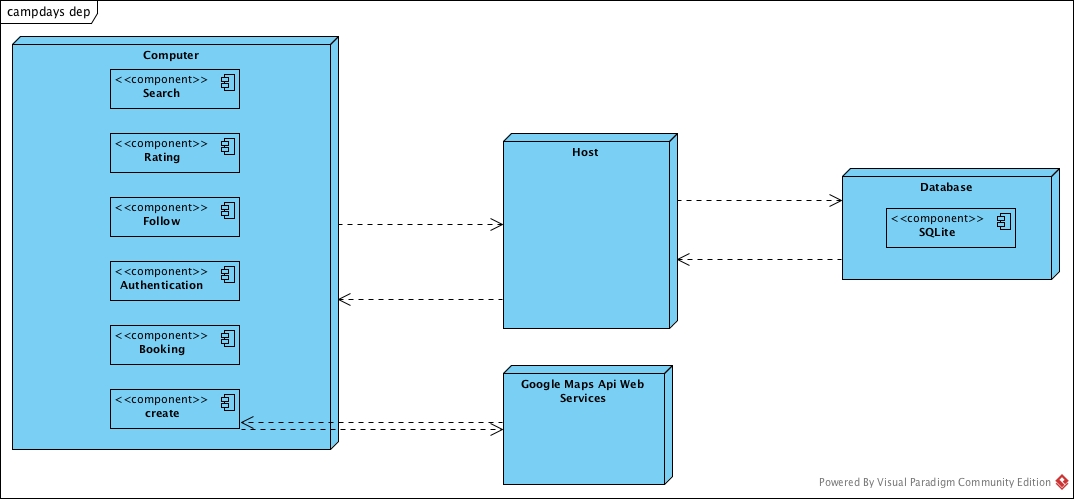
In our Campdays system, we designed subsystems based on software engineering requirements, working more efficiently and working together. This provides us a coherence. We divided to our system some subsystems that are ; add camping page interface, Admin interface , the reservation interface , maping interface , user follow interface , owner accept interface , owner register interface , the web-site main page , the main-page without sign in home page , camping detail interface and the owner home page interface.

## System Decomposition



* *Create Camp Location Subsystem:* This allows users to add camp locations to the environment.
* *Authentication Subsystem:* This subsystem is fully responsible for user login and register. Authorized users can access the system through this subsystem. Mostly campers, camp owners, adventurers.
* *Search Subsystem:* This sub-system is responsible for searching camp areas according to current location & other pre-determined parameters.
* *Booking Subsystem:* This subsystem is responsible for reservations. Users are allowed to view the intensity of camp areas and make reservations. Reservations can be cancelled also.
* *Rating Subsystem:* This subsystem is related to quality assurance of camp areas. Camp areas are rated by users. It is a great parameter for users while deciding.
* *Follow Subsystem:* This allows users to follow each other & camping areas.
* *Admin Subsystem:* This allows Admin to handle camp area creation requests. Approve & Reject.

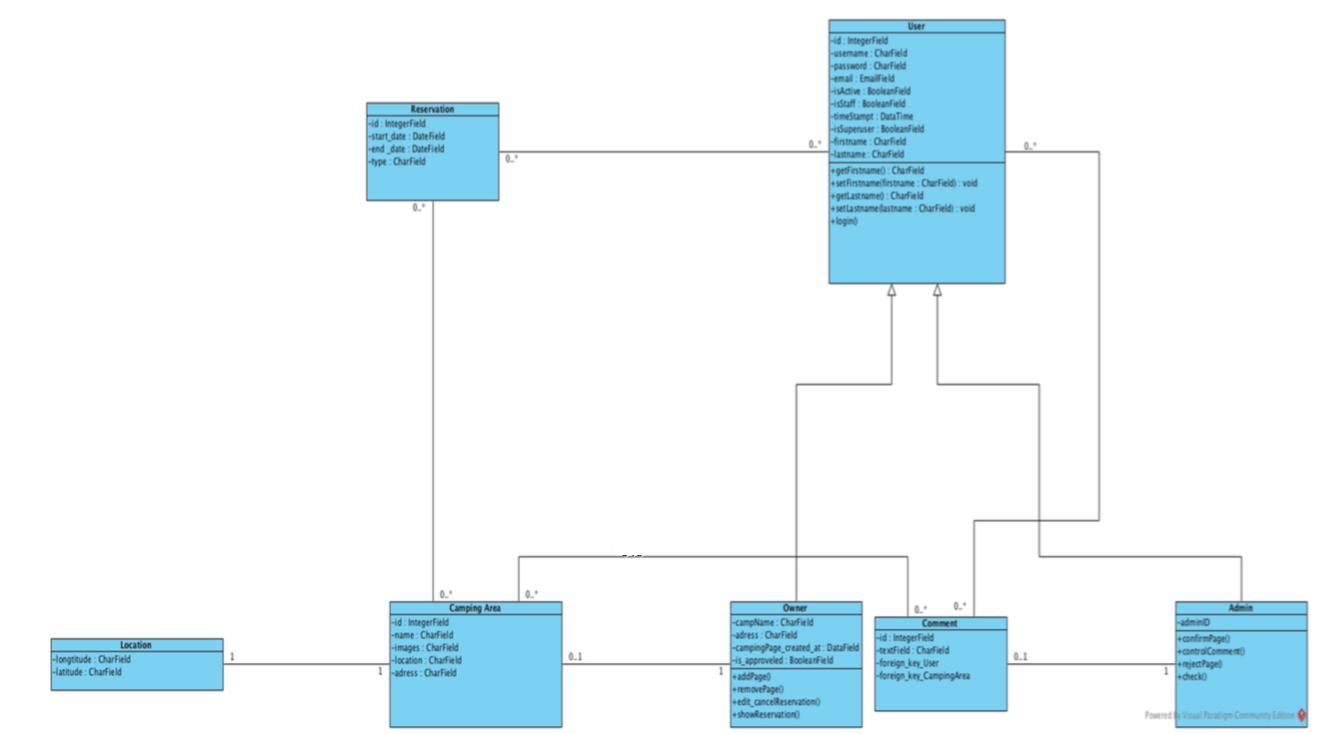
## Hardware Software Mapping



Campdays is a web application which is integrated with Google maps API web services. Creating a camp area on the system is well developed. User searches for the location on map to add the system accurately. To manage such feature, we get the text in the textbox displayed on the map and make a request at googles web services. As request is valid, request comes with some data loaded. We capture the data and set the address automatically. That’s how the system is integrate with google maps API web servers. As the form is submitted over host(localhost at the development phase), a location creation request is created and both the location and location creation request are saved to the database. As admin approves the request, location can be reached over website. This is how it works.

## Persistent Data Management

Data management of Campdays Web Application is handled by SQLite which is powerful, serverles, embedded relational database management system. Django Framework comes with builtin SQLite Database. SQLite as a self-contained, file-based database, SQLite offers an amazing set of tools to handle all sorts of data with much less constraint and ease compared to hosted, process based (server) relational databases. Here is a scheme of our database relations.



## Access Control and Security

The application is a multiuser application so it consists of 3 types of users which are camp owner, camper and administrator. Because of this, the application will provide different interfaces for each user type.

First, the administrator will connect to the system with the membership interface, and will do the administrator's duties, such as adding, updating, editing, deleting etc. By the way, registration is not necessary for the administrator through the system website, information will be entered manually into the database at the beginning of the system and the administrator will be the authority that will access the database directly. As a summary, the administrator does not have to register because it is initially registered in the database and the system with Django framework. The administrator registers the manager. Each registration process included the manager registration made by the administrator, will be done with the user interfaces of the system. The system will store all the information in the database and in the login processes again the system will use them by collecting data from the database. The information in the database will use both the confirmation and the use of the system for users. All types of users must log in to the system with their username and password.

During registration, field filling does not require access to the database, while completion of the process requires the data to be written to the database, which requires read and write access to the database. In that case, the required database fields will be blocked and simultaneous access of multiple users will be denied.

For some situation like updating or deleting information it is necessary to update one of the tables in the database in its phase of completion and therefore must be handled with more care since several users can be the cause of updating the table at the same time. This will also be avoided by blocking.

Finally, viewing the information or lists again requires read-only access to the database. Therefore, multi-user access does not impose problems and new restrictions.

As last words, the usernames and passwords of users will be stored in the user table. No one else accepts the administrator can have access to this information. Authentication interfaces are different for each type of user and will be directed to their own main pages after the login process.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Actors/  Classes | Admin | Camp Owner | User | DbConnection |
| Admin | Set\_camp\_page ()  Set\_Statistic()  Set\_User\_Account() |  |  | Login()  logout() |
| Camp Owner | Set\_camp\_page () | Set\_camp\_page ()  Show\_Reservations()  check\_feedbacks() | Follow() | Login()  logout()  Register() |
| User |  |  | Register()  SetProfile()  Show\_Location()  Search\_Location()  Booking()  Follow()  Rating() | Login()  logout()  Register() |

## Global Software Control

Our system has MVC (Model - View - Controller) software architecture. Campdays is thread safety but also multithreaded program either because our system must provide many users at the same time to order place online. And also, some functions of our system should be synchronized for providing less problems during debugging and testing but especially while website is working for real customers. But still threads became with many problems.

## Boundary Conditions

Startup: go to system URL and login

Shut Down: click log out and close browser

Error Conditions:

* Logging in:
  + Username or password field is blank.
  + Password is not 8 characters long or more.
  + Password and username don’t match.
  + Username is wrong or does not exist.
  + The welcome screen does not appear after logging in.
* User settings
  + User is unable to change certain settings or changes don’t reflect.
  + Between the time of editing and updating, the system crashes.
* Data Entry
  + The system fails when the Camp Owner is entering information.
* Camp Owner Entry
* Camp Owner informations cant be not exist.
* Camp Owner informations cant be blank.
* Camper Entry
  + Camper informations cant be not exist.
  + Camper informations cant be blank
* Reservation table:
* People capacity cant be zero.
* People capacity cant exceed the total amount of the place.

# References

The following is an example of listing a book in this section. Check the text to see how it is cross referenced (The whole document is based on [1]).

1. Bruegge B. & Dutoit A.H.. (2010). *Object-Oriented Software Engineering Using UML, Patterns, and Java*, Prentice Hall, 3rd ed.
2. <https://docs.djangoproject.com/en/2.0/ref/databases/>
3. https://www.uml-diagrams.org/composite-structure-diagrams.html