



**POLITECNICO**  
**MILANO 1863**

Software Engineering Project  
Lorenzo Amici, Marina Ranghetti,  
Marta Rossi, Yinyao Zhang

# Design Document

---

**Deliverable:** DD

**Title:** Design Document

**Authors:** Lorenzo Amici, Marina Ranghetti, Marta Rossi, Yinyao Zhang

**Version:** 1.0

**Date:** 26-May-2019

**Download page:** <https://github.com/MarinaZen/SoftwareEng.git>

**Copyright:** Copyrightc 2017, Lorenzo Amici, Marina Ranghetti, Marta Rossi, Yinyao Zhang – All rights reserved

---

## Contents

1. Introduction .....	4
1.1 Purpose.....	4
2. Structure Design.....	4
2.2 Database Design .....	4
2.2 Flask Application.....	6
2.2.1 Template .....	6
2.2.2 Python .....	8
4. Effort Spent .....	9

## 1. Introduction

The aim of the design document is to guide the development and provide high level information on the structure of the software. This document describes a set of design characteristics required for the implementation of the web application.

In details this document contains a full overview of:

- Database's structure in terms of the tables utilized
- Software's structure, such as independent programs, template and python script

### 1.1 Purpose

The goal of this project is to develop a web-app for managing bike-sharing's service information.

Bike-sharing is a short-term bicycle rental service available in different urban locations characterized by bicycle transit.

The service is accessible through applied technology (smart cards and/or mobile phone). In particular, it offers 24h/7 service.

The development of this web application has been commissioned from Comune di Milano that will eventually release privileged permissions to obtain statistical information and other utilities. In the following a more precise description is provided.

## 2. Structure Design

The structure of this web-application has been designed on two independent and distinct levels: a script in charge of creating the database and the Flask application.

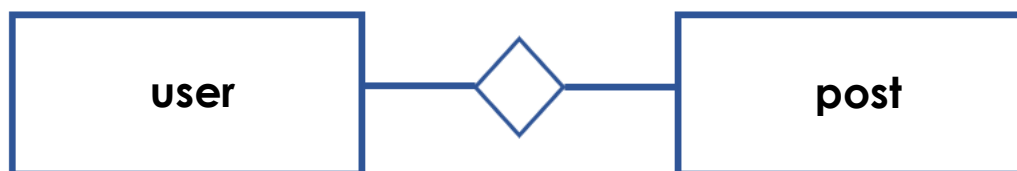
### 2.2 Database Design

The design of database largely determines the usability and robustness of the system.

**SQLAlchemy** and **PostgreSQL** -open source object-relational database- has been chosen for managing, updating and accessing information.

Two main tables were generated following a specific structure, in particular the database schema is shown below:

#### Conceptual model





### Physical model

#### **User Table**

```
CREATE TABLE User (  
id INTEGER PRIMARY KEY,  
username VARCHAR(20) UNIQUE NOT NULL,  
email VARCHAR(120) UNIQUE NOT NULL,  
password VARCHAR(60) NOT NULL  
)
```

#### **Post Table**

```
CREATE TABLE Post (  
id INTEGER PRIMARY KEY,  
title VARCHAR(100) NOT NULL,  
date_posted DATETIME NOT NULL,  
content TEXT NOT NULL,  
user_id INTEGER NOT NULL,  
FOREIGN KEY (user_id) REFERENCES user (id)  
)
```

#### **Bike Table**

```
CREATE TABLE bike (  
time TEXT,  
number of stations* DOUBLE,  
)
```

\*one column for each station (from 1 to 62).

#### **Stations Table**

```
CREATE TABLE stations (  
  id BIGINT PRIMARY KEY,  
  geom GEOMETRY(Point, 4326),  
  bike_sh VARCHAR(50),  
  indirizzo VARCHAR(50),  
  stalli BIGINT,  
  localiz VARCHAR(20),  
  lat DOUBLE,  
  lon DOUBLE,  
)
```

#### Codes Table

```
CREATE TABLE codes (  
  PIN_codes TEXT NOT NULL,  
)
```

## 2.2 Flask Application

The Flask application includes the template and Python part, both of them have their purpose, internal structure and, most important, they interact with each other.

### 2.2.1 Template

Dedicated templates were created depending on their function for the web-application.

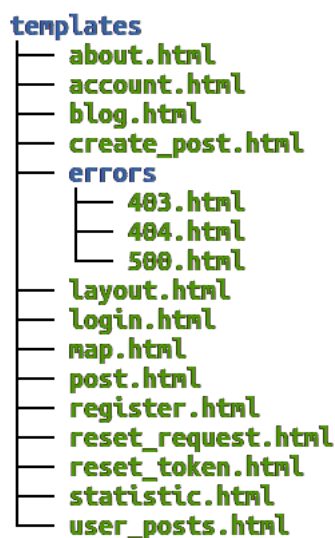


Figure 1.1: Tree structure of the templates

#### [About](#)

In this page users can find generic information about the service our web-app provides.

#### [Account](#)

Section in which the user -if registered and logged-in- can change the settings of his personal profile, such as profile picture, username and password.

#### [Blog](#)

Page available to every user in order to visualize already published posts.

#### [Create post](#)

Section in which is possible to create and publish new posts or comments. It is available only if the user is already registered and logged-in.

#### [Layout](#)

This page is not addressable, but it only defines the main structure of all the templates that have been used. The navigation bar and the CSS or JavaScript libraries utilized are declared in this page.

#### [Login](#)

Registered users are redirected to this form in order to log-in with their credentials.

#### [Map](#)

Page dedicated only to the building of the GIS system in order to visualize and interact with geospatial data. We decided to use Leaflet's libraries to manage geospatial data, and a map allows the visualization and interaction with every docking station in the database. The map can be chosen by a cluster of three, the default one is the one built using OpenStreetMap.

Leaflet's plugins allowed us to build a geocoding and routing utility. A user can localize and see his position on the map and eventually use the routing machine to navigate to a chosen docking station.

#### [Post](#)

Management of the posts of a user. Only the owner of a post can modify or delete it.

#### [Register](#)

Users are redirected to this form in order to register themselves to the service, by providing their username, email, password and a PIN code provided by Comune di Milano.

#### [Reset request](#)

Dedicated form for resetting the password.

#### [Reset token](#)

Dedicated form for inserting a new password and confirming it.

#### [Statistics](#)

Page dedicated to the visualization and interaction of the general and specific statistic of each docking station.

#### [User posts](#)

Page dedicated to the visualization of all the posts of one specific user

### Errors:

- [403](#)  
In case of permission problem, it will be shown this kind of error
- [404](#)  
When the page doesn't exist, this page will be rendered
- [500](#)  
Generic error or problem of the application itself

### 2.2.2 Python

Inside the Flask web application different modules have been distinguished, referring to: the users, the posts listed in the blog and the main pages of which the web app is composed of.

They are all placed inside the same directory in which a common constructor file (“\_\_init\_\_.py”) is used for their management. Its role is to set up instance variables, specific for each module, to have the proper initial values when objects are created.

Each module is also provided with specific routes, used to create the association between the requested URL and the function that handles it.

The “main” module decorators refer to the web app pages: Home, About, Blog, Map.

The different “posts” decorators are set to give the user the possibility to create, update, delete their posts in the blog page.

The “users” decorators are defined to allow the user to register, login, logout, update credentials, reset the password.

The forms have been implemented using specific classes; a general form is defined to create new posts and forms to do registration, login, update of the account, reset the password are implemented for the user's interaction with the web app.

Finally a “static” directory has been added within the above modules in order to better manage the pre-rendering of the templates of each page, that can be displayed to users before the JavaScript finishes loading.

The libraries used for the web app development are the following:

- ❖ **Flask version 1.0:**  
The Flask component is first of all imported from flask package in order to create the application instances needed.
- ❖ **WTForms version 2.1:**  
it's a form input as a handling and validation library for python applications which ties together the app framework, the database and templating engine to make validation workflow easier. It is not specific to flask.
- ❖ **Flask\_wtf version 0.14.2:**  
It's a flask extension that specifically makes tying WTForms into flask.



From it we also imported FlaskForm package in order to create python classes to be representative of our forms that are automatically converted in html forms within the templates.

❖ **SQLAlchemy version 1.2.7:**

It's the Python SQL toolkit and object relational mapper (ORM). Once flask\_SQLAlchemy is installed it is imported in the flask application.

❖ **Flask-SQLAlchemy version 2.3.2:**

It's a flask extension that provides useful defaults to create and configure engine, connection and session to work with the flask application.

It was used in the bike-sharing web app to create a database where user's credentials are stored in order to register and login the web app and to let them create and store in the database their posts on the blog.

❖ **Flask-Bcrypt version 0.7.1:**

It's a flask extension useful to hash passwords so that they can't be retrieved from database from external people. A hashed password can be created both in bytes or as a string.

❖ **Bcrypt version 3.1.4:**

It's the flask\_bcrypt library used to create an instance for this class. It allows to generate passwords each time with a different hash (even when using the same password), so that if someone steals it from the database he/she wouldn't be able to use a hash table to crack it.

❖ **Flask-Login version 0.4.1:**

It's a Flask extension used to put a validation check to verify if the new account is already existing in the database created for user's credentials.

## 4. Effort Spent

Marina Ranghetti: RASD, DD and web application (website and map)

Marta Rossi: RASD, DD and web application (website and code comments)

Lorenzo Amici: RASD, DD, statistics and their implementation

Yinyao Zhang: DD, testing of use cases and code comments