RoadFix project by Haakon Flaten - Paal Kjekstad

POLITECNICO
MILANO 1863

Test Plan Document

Deliverable:	DD
Title:	Design Document
Authors:	Haakon Flaten and Paal Kjekstad
Version:	1.0
Date:	2-Mai-2021
Download page:	GitHub repository
Copyright:	Copyright © 2021, H.Flaten and P.Kjekstad – All rights reserved

Contents

1 Introduction

1.1 Purpose

This document will include information that is more on technical side of the project then the RASD document. A full overview of the program and how the different parts of the software works and interact with each other. The RASD document are been referred to throughout this document. This document and the RASD document gives a complete explanation of the program.

1.2 Structure of the database

The Epicollect5 file has information that describes potholes in England. The dataset has 190 items which has 13 attributes. Epicollect5 is a web page where information can be added at any time to the dataset, so the 190 items can increase as the project drags on. Not all of the 13 attributes will be used. The once focused on in this project is the longitude, latitude, name of the damage and the URL to the picture of the damage. This will be sufficient enough to achieve the goals explained in the RASD document. Not all the 190 items has all this information and will therefore be removed from the dataset.

The data will be taken straight from Epicollect5 by using requests package in Python. That way it makes it possible to use information that gets updated through Epicollect5 consecutively.

1.3 Reference Document

The assignment document: AY20-21 Project.pdf

2 Software structure

2.1 Flask

Flask makes it possible to create servers in python. Flask is a microframework that is used for minimalistic web application framework. Importing from flask: Flask, request, render_template, redirect, flash, url_for, session, g

- Session: create session were users can delete and store data
- Request: HTTP library in python
- Flash: easy to make comments to the user, for example: "Incorrect password/username"
- Render_template: render html pages
- Redirect: the user gets redirected to correct page. Normally used with url_for

2.2 JSON

JSON (JavaScript Object Notation) is a flexible data exchange format for REST API. It is a data interchange format that uses human-readable text to transmit and store objects consisting of arrays and attribute-value pairs. We import raw-data, json.loads, response and pdf.json_normalize.

2.3 Base.html

The base html file is being used in most of the other html files. The main purpose of this is to get the same design on all the different pages in our program. It also includes links which makes it possible to move easily from page to page on the web application. The style used is described through a css document. When the base is used for a foundation to the html file you get a equal layup of some parts of the web page related to the html file. The first part that is similar to all the pages is that it got a navigation bar in the top of the page. This bar includes the name of the program (RoadFix) and two links to different pages. If you are logged in the bar while give you the possibility to logout. If you have not jet started your session you will get the possibility to login or to register a new user. The name RoadFix in the navigation bar is a hyperlink to the home page.

2.4 Home.html

The home page is where you find the map and hyperlinks to "workspace". It is based on the "base.html" file and will have the navigation bar on the top of the page. It will include the logout hyperlink that sends the user back to the login page. The page will also have two hyperlinks that gives the user a possibility to go to the "workspace". The map will have all the rated damages drawn on it, as explained in the RASD document. The URL key to get to the homepage is ('/home').

2.5 Workspace.html

This file is using the base html as a foundation. The page gives out the link to the picture the user needs to rate. Under the link to the picture is the rating form. The user will type in the rating in this form. The URL key to get to the workspace is ('/workspace'). After completing the rating of a damage you get redirected back to the workspace with an updated link to a new picture.

2.6 Index.html

The index html is making a hyperlink in the home page. These hyperlinks directs the user to the "workspace". This button is available only when you are logged in. This html file uses the base as a foundation. The index html file does not have a URL key.

2.7 Register.html

This page registers your personal username and password and stores it in the programs database. This information needs to be filled in to two forms. Above the forms are the title "Register" and under the forms are a button with the text "Register", which send you to the login page if the both forms are filled out. If not a message will appear on the screen explaining the problem. The username and password will be saved in the WSGI server and can be used to login. The navigation bar in the top of the page has two links, register and login. The URL key to get to the register page is ('/register'). After making the username and password and pressing the button below the forms the program redirects you to the login page.

2.8 Login.html

This page lets you login with the user you have made. The page has the same layup as the register page except that the title above the the two forms are named "Log In" and the button bellow the forms are named "Log In" as well. The URL key to get to the login page is ('/login'). After typing in the username and password and pressing the "Log in" button the program redirects you to the homepage ('/home').

2.9 Update.html

The update page lets the user edit a damage that is already rated. The page has the foundation of the base html file. The hyperlink to the picture of the damage is visible under the navigation bar. Below that is the window where the user can edit the old rating. It is also two buttons on the page. One that saves changes done to the information stored before and one to delete the damages. When the user presses the button that saves the changes or delete the damage the program will send the user back to the home page. The URL key to this page is divided into two. The first part is the number the damage is saved as and the other part is ('/update'). For example: For damage-2 the URL key is ('/2/update')

3 Use cases

3.1 UC1: Register user

@app.route('/register', methods=('GET', 'POST')).
It is two HTTP requests the function needs to answer:

- If the request is POST:
    1. Get the information sent by the user (username and password)
    2. Must see if username and/or password follows the defined syntax
    3. See if the username is in the database
    4. If username exists, flash a message that tells the user that the username exists
    5. else, the username and password is saved to the database and flash a message that registration was successful
    6. Redirect the user to login page
- If the request is GET the user is redirected to register page

3.2 UC2: Login case

@app.route('/login', methods=('GET', 'POST')).
It is two HTTP requests the function needs to answer:

- If the request is POST:
    1. Get the information sent by the user (username and password)
    2. Must see if username exists in the database
    3. If not, flash a message that tells the user that the username was incorrect
    4. If username exists, see if password is correct
    5. Correct password and the user is logged in and the user-id is stored in the session variable
    6. Wrong password and a flashed message tells the user that password was incorrect
- If the request is GET the user is redirected to login page

3.3 UC3: Logout case

@app.route('/logout'). The function have to be able to:

1. Delete the session variable
2. Redirect the user to the login page

3.4 UC4: Rate damage

@app.route('/workspace', methods=('GET', 'POST')).
It is two HTTP requests the function needs to answer:

- If the request is POST:
    1. Upload data from EpiCollect5
    2. Get the information sent by the user (Rating)

3. Must see if the rating is between 1-10.
4. if not, flash a message that tells the user that the rating is invalid.
5. if correct, store the rating in the database together with the position of the picture from EpiCollect5. A message tells the user that the rating was saved and the workspace is updated

- If the request is GET the user is redirected to workspace

3.5 UC5: Coloring of damages

@app.route('/') @app.route('/home).

- Takes in the information about the position and rating of the damages from the database
- Defines which damages are yellow, orange or red.
- Plots with help of the matplotlib and contextily packages a picture of the damages on a map
- Draws that map on the home page ('/home')

3.6 UC6: Remove rating

@app.route('/<int:id>/update', methods=('GET', 'POST'))
The information to every damages is stored and get a own URL key.

- If the request is POST:
  1. The user adds new information to the damage
  2. Must see if the rating is between 1-10 or registered as fixed
  3. if not, flash a message that tells the user that the rating is invalid
  4. The data stored about the damage is edited
  5. The app redirects you to the home page and the map there is updated with the new information
- If the request is GET the user is redirected to update page

3.7 UC7 and UC8

Both use cases 7 and 8 are taking the whole program to use. So they will reuse the explained use cases above. Therefore, they will not be explained in the design document.

4 Appendix

5 Software used

- Overleaf as an editor for Latex

6 Effort

Both have used the same amount of work and have helped each other with all sections. There is no point to specify hours spent on each activity since our work overlaps.

- Paal Kjekstad: 10 hours
- Haakon Flaten: 10 hours