POLITECNICO
MILANO 1863

Design Document and Test Plan
Version 2.0

Web-Based Application for Water quality monitoring of the Pomperaug River

Authors

Ahmed Omer Ahmed Mukhtar – 10833977
Huzaifa Mohammed Khair Khider Abdulaziz - 10782594
Forough Rajabi - 10807901
Madeleine Abbes - 10859693

Table of Contents

Introduction

DESIGN DOCUMENT

Design clarification and documentation is a significant Phase in the development of a software system. It follows the requirement analysis stage and demonstrates how solutions to the formerly identified client needs shall be implemented. The purpose of this document is to elaborate on the software design and test plan of a web-based application dedicated to the communication and visualisation of the citizen science data collected in the context of the Pomperaug watershed communities water quality monitoring, which has been conducted by The Pomperaug River Watershed Coalition (PRWC) in west central Connecticut, United States.

The Software Design and Test Plan document primarily aims at providing guidance to the development team by outlining the system's overall architecture, illustrating the workflow of what needs to be built and how, and clarifying the relationship and connection between different software components. Although it is a technical document serving a blueprint of the software's code, not necessarily aimed at the stakeholders, the client party can use it as well to better understand the underlying technology of the product.

PRODUCT DESCRIPTION

The water quality monitoring web-application demands implementation as a dynamic website, since the majority of page components require data visualisation/customization that exceeds the abilities of static web pages. To support the development of such a website, the use of a software framework can be helpful, since it provides libraries for automations such as templating engines or session management, together with predefined classes or functions that can be used to process user inputs or interact with databases. Furthermore, as specified in the RASD, the application shall be developed in python, therefore demanding a development framework that is compliant with WSGI, which is the specified protocol that describes how a web server communicates with web applications written in python.
For these reasons, the Flask framework has been chosen to support the development of the water quality monitoring web-application. It is one of the most popular WSGI microframeworks used for web-application development with python, since it is simple yet extensible. It depends on the Jinja template engine, which allows the generation of dynamic html pages, the Werkzeug toolkit, needed to write WSGI-compatible applications in python, and it does not prescribe a database backend, therefore preserving the system's flexibility. Essentially, Flask provides all the means necessary to meet the project's requirements.

Project Database

Epicollect5 dataset

The web-application displays data that has been gathered and stored on Epicollect5 the dataset contains a total of 179 entries (last visited 30-oct-2020). The system will comprise its own database, that the web-application will make use of. Not only does this intermediate storage of data decrease waiting times it moreover increases the system's reliability, since it is then independent from the Epicollect5 server and can uphold service even when it fails.
The Epicollect5 raw data will be retrieved through the Epicollect5 API, and as mentioned in the RASD document Epicollect5 raw data has been collected via the same questionnaire, resulting in data entries containing 24 attributes each. Some of these attribute's requests to upload photos or to mention the used method of sampling, etc. thus it will be pre-processed, filtered according to the project's interest, and stored in the project's inherent database.

PostgreSQL Database

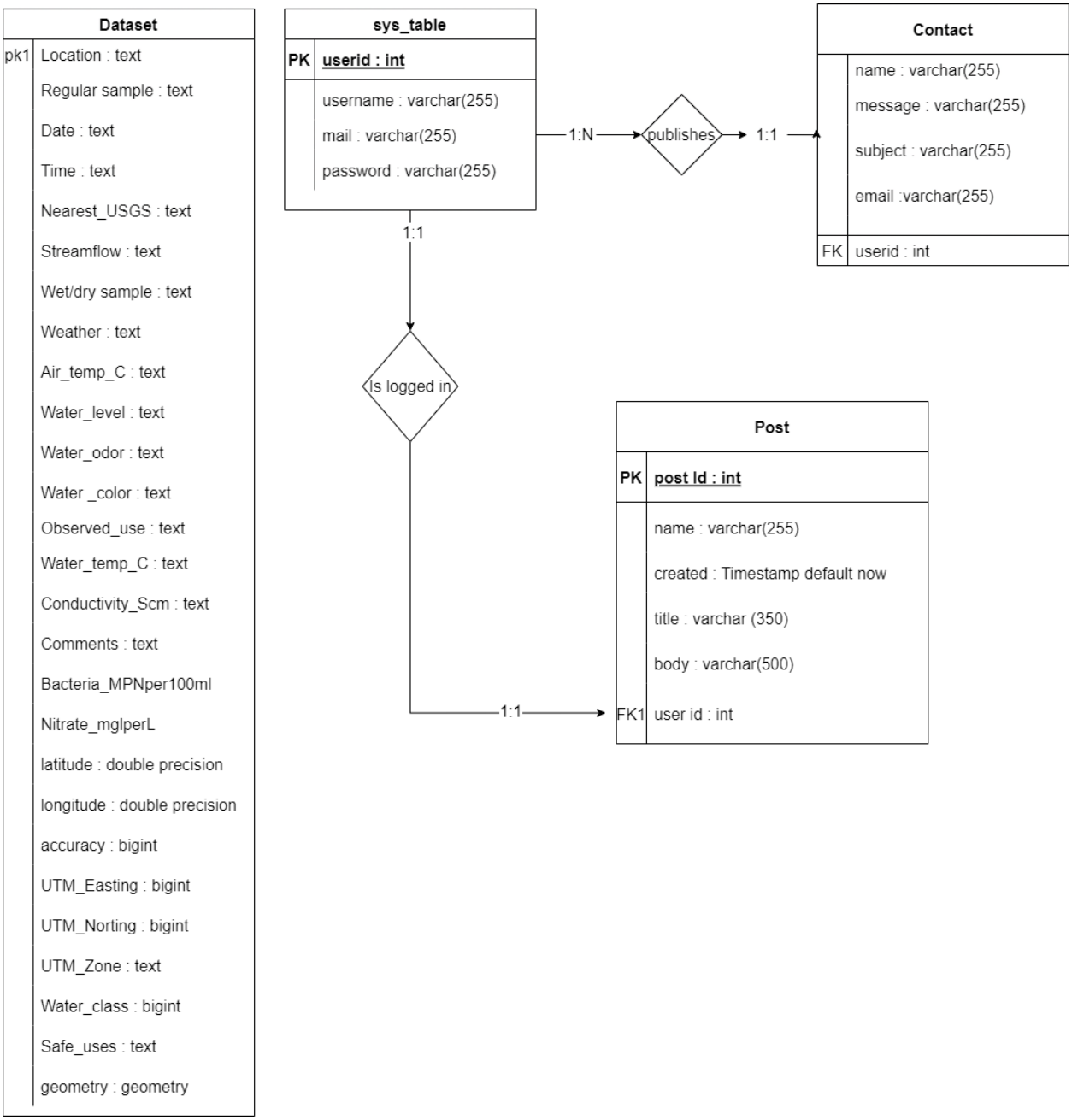
The extracted dataset should be stored on a database server that can interact with the web application. PostgreSQL has been chosen as a Database management system since it is both a DBMS and inherently contains a database server with a local database on which the project's data will be stored.

The project database will contain:

- A table containing the processed dataset.
- A table to store members' credentials.
- A table connected to both the geodataframe and members login information, saving the members' messages.

Architecture of the database

The diagram below describes the organisation of our database, with the relationship between them. Each table of the diagram represents a table in the database, with its attributes. The diamond linking two tables, gives the link between the tables.



System Architecture

System architecture is the conceptual model of our piece of software defines its organization structure and behaviour in terms of components and connectors.
The system architecture will be developed in the following way, using several scales and points of views

Component diagram - Architecture in the large

The system in large consists of a three main components:

Web sever

A several HTML, CSS and java script files saved in folders in the localhost. These files creates the web pages of the system, and rendered by the application server (WSGI).

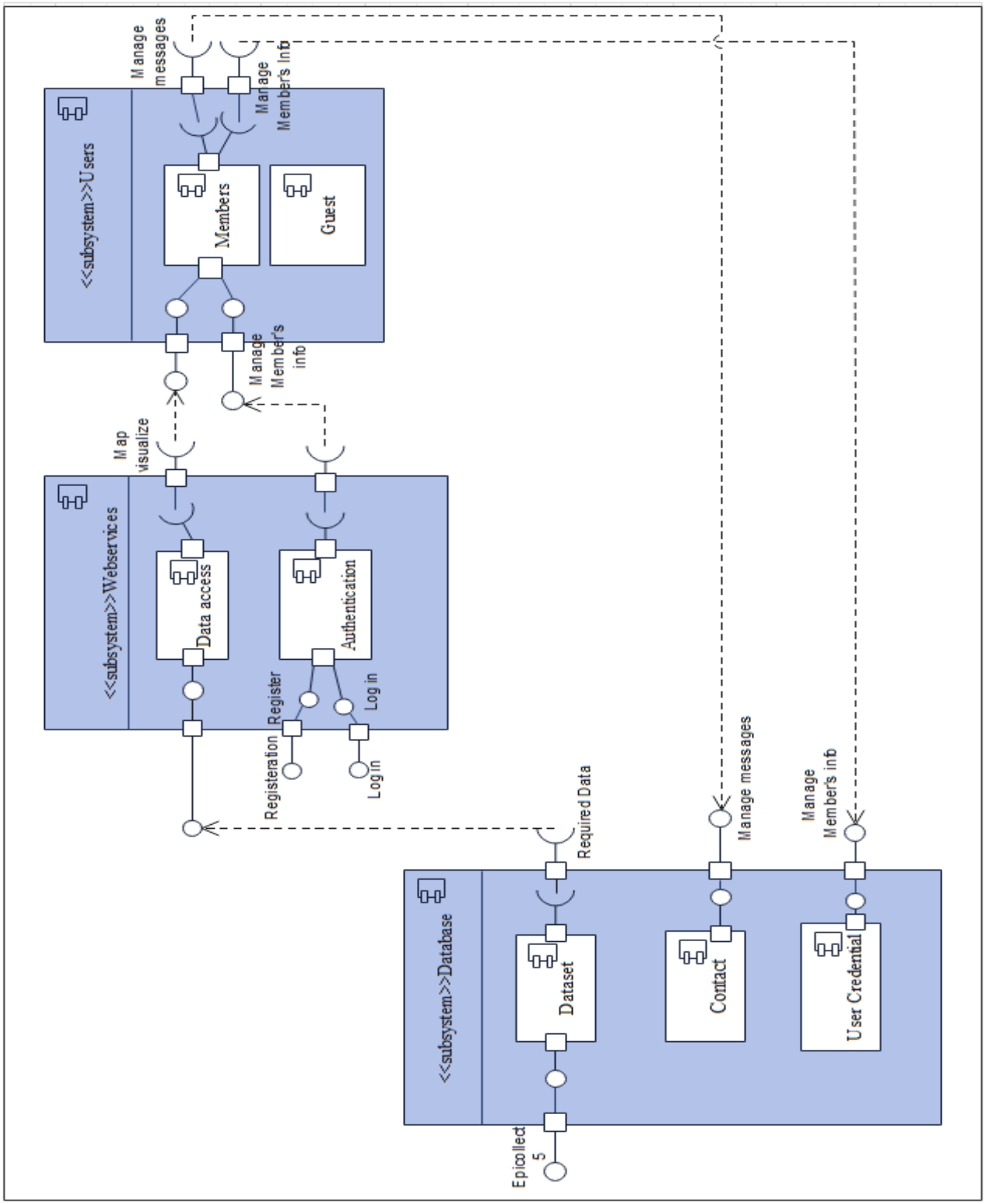
Application server (WSGI)

WSGI is an acronym of (Web Server Gateway Interface), it is the specified protocol that describes how a web server communicates with web applications written in python, application server is running by Flask framework. With additional libraries, which are the Werkzeug toolkit that writes WSGI-compatible applications in python, and Jinja template engine, which allows the generation of dynamic html pages. These entire components are managing users' sessions, connecting with database and generate dynamic web pages. Also a Bokeh library to plot the map and show its attributes in the HTML interface.

Database

For the system database, PostgreSQL has been chosen as a Database management system since it is both a DBMS and inherently contains a database server with a local database. the software database will contain a table containing the processed dataset of Epicollect5 data, table to store members' credentials (username, email, password) and a table connected to both the geodataframe and members login information, saving the members' messages.

The following diagram describes the organisation and wiring of the physical components of our system.


Manage messages
Manage Member's info
<<subsystem>>Users
Members
Guest
Manage Member's info
Map visualize
<<subsystem>>Webservices
Data access
Authentication
Register
Log in
Registeration
Log in
Required Data
Manage messages
Manage Member's info
<<subsystem>>Database
Dataset
Contact
User Credential
Epicollect 5

Architecture of the code

WSGI-COMPLIANT WEB SERVER

The web server needs to be WSGI compliant. This means that it must understand that HTTP client requests can be mapped to both static sources (HTML) as well as to sources that are dynamic i.e., to python code. The HTTP requests concerning static HTML are still held and served by the webserver itself. even when it is WSGI-compliant, but it will forward the HTTP requests for dynamic sources to a WSGI-server, that runs the python application, which will also return an HTML, which can then be served to the client by the webserver just like a static HTML would be. The webserver is also holding and serving the CSS files, referenced by the HTML responses

One HTML template is defined for every page of the web-application and is rendered by the Jinja template engine on HTTP request.

REGISTRATION FUNCTION

The function must answer to two both get and post HTTP requests. The user is prompted for information input (username, password, email) an the registration.html page, that is invoked by the HTTP get request. The user input is passed to the registration function as arguments, the function checks the user data frame (database) for already existing data associated with the provided email. If it finds a match, it returns an error message, if it does not it writes the user input to the user data frame of the database and redirects the user to the login.html page.

LOGIN FUNCTION

The login function operates with the same principle. It must answer to both get and post HTTP requests. The get HTTP request calls the function that returns the login.html page. On this page the user is asked to provide username and password, which are passed to the login function as arguments by a post HTTP request. The function checks if all necessary user input was provided and then reviews the user data frame in the database. If the function can match the provided user input to an entry of the data frame, it returns the base.html page (which the user is thus redirected to). If it does not find a match, it returns an error message.

LOGOUT FUNCTION

This function clears the user session variable and redirects them to the application’s start page “start.html”

MAP DISPLAY

The map display retrieves the dataset table from the database, transforms it into a web map. The dataset is retrieved from the database and transformed into a GeoDataFrame. Data is then pre-processed in order to have everything ready for the map (colour mapper, x and y coordinates). Data is then displayed on the CartoDBpositron basemap, with different colours as a function of the bacteria concentration in the point. Other important information, like the date of acquisition of the sample, are displayed when the user hovers over the points.

CONTACT FUNCTION

This function answer to two HTTP requests (Get, post). The user is prompted for information input (name, email, subject, message) in the Contact_us.html page. that is invoked by the HTTP get request. The user input is passed to the Contact function as arguments, the function checks the user email in the (database) for already existing usrerid associated with the same email. If it finds a match and there is no already saved message in Contact table in the database, it returns a message that inform the user that they send there message successfully. if it does not it retunes a message asking user to register first. if it finds a match and there is already saved message in Contact table in the database, is return a message informs the user that they already sent a message.

JINJA TEMPLATE ENGINE

The Jinja template engine renders the dynamic HTML pages using basic python concepts such as variables, loops and lists resulting in a customized HTML file. The web-application displays content with three static HTML pages, concerning user registration and login (1,2,3) and four dynamic HTML templates (4,5,6,7) for the main pages.

1. **Start.html:**
   Visualises the options to either sign in or sign up or enter as a guest to the system.
2. **Signup.html**
   Contains all fields necessary for user registration to the system.
3. **Login.html:**
   Contains all fields necessary for registered members to login.
4. **Home.html:**
   Provides information about the watershed, the type of bacteria under study, and displays an interactive map, that changes upon HTTP request and a user-friendly filter interface.
5. **Generic.html**
   Provides elaborated information about the stream monitoring.

6. **About_us.html:**
   Provides contact information and a biography of the development team members.
7. **Contact.html:**
   Contains all fields necessary for registered members to send messages.

The HTML templates reference specific CSS files that dictate the style and appearance of the web page, including component colours, text alignment and border styles etc. The CSS files are held and served to the client by the web server.

USE CASES APPLICATIONS AND IMPLEMENTED REQUIREMENTS

To explain the functionalities of the software, the interactions between the components and possible exceptions, this section provides an explanation of the actions performed by the software and the user in a list of cases that are useful to explain the internal processes of the application.
This section describes what happens on the server and client side when the user cases occur by indicating the different actions that take place in these situations.

UC1: Registration:

- signup.html page
- The user is prompted to provide personal information such as username, password, and email.
- The registration function checks that the entered inputs does not match any other inputs in the user information data frame on PostgreSQL.
- If it does the software will provide an error message to the user “Username/e-mail already exists”
- If it does not, the registration function stores the user input (username and password) in the user data frame.
- The user is redirected to the login.html page.

UC2: Login:

- Login.html page
- The member is asked to provide his/her username and password.
- The login function receives this input and compares it to the information stored in the PostgreSQL user information data frame.
- If the member’s login information matches the ones in the database, the function redirects the user to the base.html page.
- If it does not the software returns with” wrong username or wrong password”.
- The member got the ability to reset the password.

UC3: Map view

- home.html
- The user registers to be able to see the map.

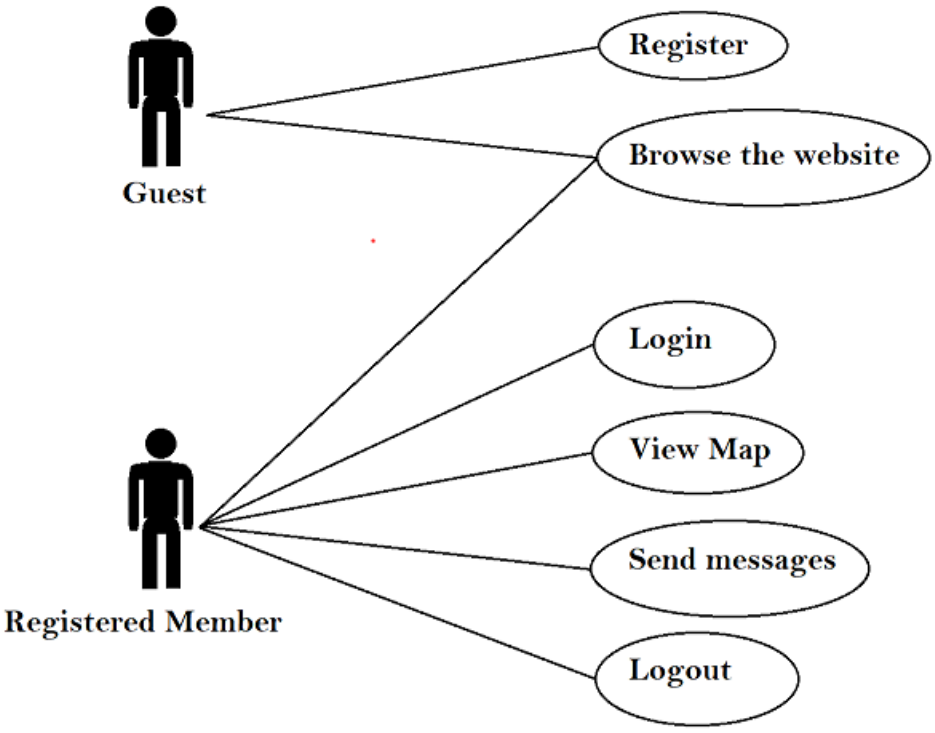
- The hover and zoom tools allow the user to visualise all of the data points, and to have information from one particular acquisition they are interested in (depending on location, acquisition date and bacteria level)
- The mapping function plots the array of coordinates on a base map and passes the plot to the jinja template engine, which renders the base.html using the provided variable.

UC4: Contact

- ContactUs.html
- Only members got the ability to send messages about their impressions on the software
- The function provides a field for adding comments or a title about the our software
- The function stores and displays members' usernames and their posted messages in the contact table on the database.
- if the end user is not a registered member the system will ask them to the registration page

UC5: Logout:

- The member is logged in from login.html.
- A logout botton will appear in home.html.
- The logout function clear the session after user clicks on the button of logout.


Register
Browse the website
Guest
Login
View Map
Send messages
Logout
Registered Member

Testing plan

Here is a list of Black-box test cases defined for the use cases of the software depending on (inputs, hypotheses on the internal state, outputs).

UC1: Signup:

Reference use case identifier	UC1
Test case identifier	TC1
Inputs	The user access to signup page and inserts: username, email, password
Hypothesis on the internal state	➢ The inserted username already exists in the database. ➢ The inserted email already exists in the database. ➢ The user forgot to fill one or more required fields.
Expected results	➢ The system informs the user that the username already exists. Try another one. ➢ The system informs the user that the email already exists ➢ The system informs the user to fill out this field. ➢ The system redirects the user to the login page.

UC2: Login:

Reference use case identifier	UC2
Test case identifier	TC2
Inputs	The user access to login page or redirected by signup page and inserts: username, password
Hypothesis on the internal state	➢ The user forgot to fill one or more required information. ➢ The inserted username and password already exist in the database. ➢ The inserted username does not exist in the database. ➢ The inserted password does not exist in the database.
Expected results	➢ The system informs the user that to fill out this field. ➢ The system redirects the user to the home page. ➢ The system informs the user that username is incorrect. ➢ The system informs the user that the password is incorrect.

UC4: Contact:

Reference use case identifier	UC4
Test case identifier	TC4
Inputs	The user access contact page and inserts: name, email, subject, message
Hypothesis on the internal state	➢ User inserted a wrong email. ➢ User has already wrote a message and it is in the database ➢ The user inserted their message.
Expected results	➢ The system informs the user to register before they can send a message. ➢ The system informs the user that they already sent a message. ➢ The system displays a message to thank the user for contacting.

UC5: logout:

Reference use case identifier	UC5
Test case identifier	TC5
Inputs	The user accesses home page
Hypothesis on the internal state	➢ The user accesses the home page as a guest. ➢ The user accesses the home page after logging in.
Expected results	➢ The system shows (login/Register) button in the home page. ➢ The system shows (welcome + username) and logout button in the home page.