Veritas University Abuja
(The Catholic University of Nigeria)

Implementing a REST API for a Description of all Countries in the World Using Vue.js to Define the User Interface

Ata, Chinonso Anita
VUG/CSC/17/1914

Submitted to

The Department of Computer and Information Technology College of Natural and Applied Sciences

In Partial Fulfillment of the Requirement of Bachelors Degree of Computer Science

December 4, 2020

Certification

This is to certify that this project entitled “Implementing a REST API for a Description of all Countries in the World Using Vue.js to Define the User Interface” was carried out by Ata, Chinonso Anita with matriculation number VUG/CSC/17/1914 in the Faculty of Natural and Applied Science, Veritas University, Abuja for the award of Bachelor of Science in Computer Science.

Student	Sign and Date
Departmental SIWES Coordinator	Sign and Date
Head of Department	Sign and Date

Dedication

This Mini-Project is dedicated to myself for all the hard work I put in from the beginning to the completion of the project and the writing of the report and to God for giving me the strength to take on the project and for blessing me with the knowledge to complete it.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

In the world today, having facts at your fingertips is very useful and important. With search engines like Google, life has been made easier as we do not have to visit libraries or spend money on books to get the most basic facts or knowledge. We now have all the information in the world in one place, and it is easy to gain access to it by just typing in a keyword or two, and we have what we want.

What this application will do is to take this functionality further by putting all the countries in the world and their basic information such as their capital, population, continent, etc. in one place. Just like you would have in google, you could just go to the search box and type the name of any country in the world and you get information about that country that you're looking for.

As Technology is being incorporated in our everyday lives and things are being made easier each day it shouldn't stop us from still looking for ways to improve on these technologies to further make things easier in order to keep the world moving forward.

1.2 Aims and Objectives

By having a new interactive system that consists of a layout that displays all the countries in the world, the system can help users who just need quick and simple information about any country. It can also be used by students for assignments related to the subject and it can also be a fun way to learn about different countries in the world.

Thus, the application aims to produce a simple, interesting, and easy to use application that the user can refer to and rely on at anytime to give them the basic facts about any country in the world.

The objectives for developing the Rest Countries Application are as follows:

- To design a simple system and interface that can easily be viewed by any user to search for any country in the world and view simple information about the country that was searched for.

- To develop a system that is accessible anywhere and on any device.

1.3 Features of the System

The project is intended to produce an interactive system so that the user can feel interested to use the system. This is achieved by implementing interactivity especially in the design of the system, a Graphical User Interface (GUI), and responsiveness.

1.3.1 Interactivity

The definition of interaction is quite broad. (Aoki, 2000) stated that interactivity of a medium refers to a characteristic of communication settings a medium can create that allows users to interact.

Throughout the process of interaction design, the developer must be aware of key aspects in their design that influence emotional response in target users. The need for products to convey positive emotions and avoid negative ones is critical to any product success. These aspects include positive, negative, motivational, learning, creative, social and persuasive influences. A method that can be used to convey such aspects is the use of expressive interfaces. (Kamari, 2011)

In software, the use of dynamic icons, animations and sound can help communicate a state of operation which in turn will create a sense of interactivity. Interface aspects such as fonts, colour palette, and graphical layouts can also influence an interface's perceived effectiveness.

1.3.2 Graphical User Interface

A graphical user interface (GUI) is a type of user interface item that allows people to interact with programs in more ways than typing. A GUI offers graphical icons, and visual indicators, as opposed to text-based interfaces, typed command labels or text navigation to fully represent the information and actions available to a user. The actions are usually performed through direct manipulation of the graphical elements.

There are several principles that need to be considered when dealing with a GUI.

- Layout
  The interface should be a series of areas on the screen that are used consistently for different purposes.
- Content Awareness
  Users should always be aware of where they are in the system and what information is being displayed.
- Aesthetic
  Interface should be functional and inviting to users through careful use of white space, colours, and fonts. In this project for example, there will be an option for the user to change the colour theme from light mode to dark mode and vice versa which makes the user feel like they are in control.
- User Experience
  The interface should be built in such a way that it is both easy to use and easy to learn. Novice users or infrequent users of software will prefer ease of learning and frequent users will prefer ease of use.
- Consistency
  Consistency in interface design enables users to predict what will happen before they perform a function. It is one of the important elements in ease of learning, ease of use, and aesthetic.

1.3.3 Responsiveness

The system is a full web application and this enables the system to be viewed anywhere no matter the device and the design is also fully responsive so the layout is still engaging whether the application is viewed on a small device or a very large device like a television.

Chapter 2

Methodology

2.1 Introduction

The system will be implemented using the waterfall model of system development life cycle as the project methodology. This methodology is selected because of its advantage which allows the requirements to be specified at the start of the project and for proper documentation.

The Waterfall Methodology is a linear approach to software development. It is also known as the 'Traditional Approach' because it is time-tested and easy to understand.(Adenowo & Adenowo, 2013). The waterfall methodology breaks up the software development projects into steps: planning and analysis (Requirement Analysis), design and implementation, testing, system deployment, and maintenance. See Figure 2.1 for an illustration.

2.2 Project Activities

2.2.1 Requirement Analysis

In this phase, all the possible requirements of the system such as functional requirements, programming tools to be used, feasibility, and scope are captured and documented.(Petersen et al., 2009). The overall project is intended to come out with an interactive system that lets the user search for any country in the world

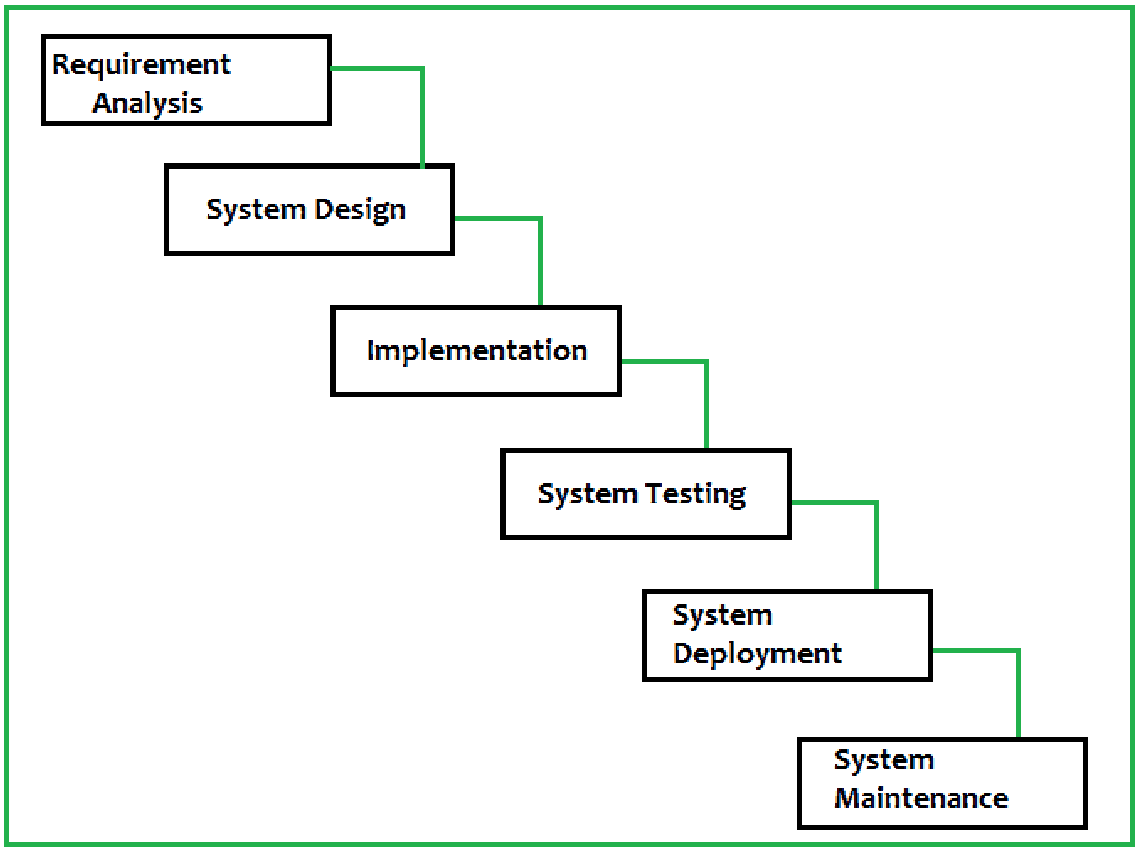


Figure 2.1: Waterfall Methodology

and get information about that country.

At the beginning of the requirement's analysis phase, the first thing that has been discussed is the programming tool that will be used which is Vue.js for the client-side, Node.js with Express for the server-side and MongoDB for the database.

2.2.2 Design and Implementation

In the design phase, the actual database or file structure, user interface, system inputs and outputs is designed. Thereafter, the actual development of the system takes place. A basic unit test is also conducted to verify that each component meets its requirement before handing the developed code over to the testing phase.

2.2.3 Testing

In this phase the system integration is tested regarding quality and functional aspects. In this case, the system will be tested by different users on their devices to make sure everything is working as it should. Furthermore, any response will be taken into consideration to improve the system in the future.

2.2.4 System Deployment

In this phase when the application has been fully tested, the whole system will be transferred from development mode to build mode. The system will then be brought into shippable state. The database will be set up on MongoDB Atlas and the application itself will be deployed to Heroku.

2.2.5 Maintenance

After the product has been released, there might still be some problems in the user environment. Fixing those issues will require regular maintenance and patches to be released.

Chapter 3

System Analysis

3.1 Introduction

The term *analysis* refers to breaking a whole into parts with the goal of understanding the parts' nature, function, and interrelationships. The purpose is to give a clear picture of the system in terms of capability required and what the software system is required to do. (Dennis et al., 2009)

System Analysis is a method of problem-solving that deals with the breaking down of a system into component parts in order to study how well the individual parts work and interact to achieve their purpose.

The basic process of system analysis involves three steps:

- Understand the existing situation
- Identify improvements
- Define the requirements for the new system.

3.2 Requirement Analysis

A requirement is simply a statement of what the system must do or what characteristics it needs to have. During a systems' development project, what the users need to do (user requirements); what the software should do (functional requirements); characteristics the system should have (non-functional requirements); and how the system should be built (system requirements).

This section includes the requirements of the system that are categorized into user requirements, functional requirements and non-functional requirements.

3.2.1 User Requirements

User requirements are the users' expectation from the system as well as the characteristics it possesses in order to fully, effectively and efficiently interact with the system. These requirements are as follows:

- The system should be user-friendly and interactive.
- The system should be accessible on all platforms.
- The system should allow the administrator to authenticate and validate user information by comparing the credentials that the user has provided and what is existing in the database.
- The system should be secured by passwords.
- The system should be able to display information about all the countries in the world.

3.2.2 Functional Requirements

Functional requirements capture the intended services, functions or tasks that the system provides, and they include the following:

- The system should authenticate users by allowing only users with correct username and password to access the system.
- The system should be able to register users with detailed information and create accounts for the users.
- With this system, the user should be able to view information about any country in the world.

3.2.3 Non-Functional Requirements

These are requirements that are not directly concerned with the specific behaviour of the system but rather the criteria that can be used to judge the operation of the system and these include:

- Maintainability of the system is easy and cheap to maintain and work with.
- Accessibility of the system is guaranteed to only authorised users by use of passwords and username.
- The system should operate on all platforms and on any device.
- The system is portable and lightweight and does not use large storage space as well having no impact on the platform performance.
- The system interface is simple and interactive.

3.3 High-level Constituent Parts

3.3.1 Database Management

The database will have the following characteristics:

- The Database will be accessible to the software.
- The Database will allow the users to store their information.
- The Database will enable user data to be queried from the database.

3.3.2 API Management

- The API will be accessible by the software.
- The API will allow users to search for data.
- The API will allow users to view information about the data searched for.

3.3.3 Software Management

- The software will be accessible from all platforms.
- The software will be able to add data to the database.
- The software will be able to fetch data from the API.
- The software will be able to display data gotten from API.

3.4 Technology to be Used

It is recommended to use a Web-based technology for the system. The advantage of using web-based technologies is listed below:

- It is easy to use (with user-friendly interfaces)
- It is free (doesn't require any license)
- It is cheaper
- It is easier to manage and maintain
- It can be accessible on any device.

Chapter 4

System Design

4.1 Introduction

The purpose of system analysis is to discover the business needs and requirements including functional requirements and non-functional requirements while the purpose of system design is to decide how the new system will operate.

System design consists of design activities that produce system specifications which satisfy the functional requirements that have been developed in the system analysis process. System design is basically the structural implementation of system analysis. A successful design builds on what was learned or gathered from the system analysis and leads to smooth implementation by creating a clear plan of what needs to be done.

The system design determines the overall system architecture which consists of a set of physical processing components, hardware, software, people, and the communication among them that will satisfy the system's essential requirements. The various procedure of usage of the new system is given here, i.e. how to, what to and on what the system will be used on. The importance of the design is to enable the system designer or researcher to know the cost of consequence of the product on the user and developer. In that the effectiveness of the system will not be obsolete.

In this section the following tools were used to describe the system: context diagram and data flow diagram, flow chart and entity relationship diagrams.

4.2 Data Flow Diagram (DFD)

Data flow diagramming is a technique that diagrams the systems processes and the data that pass among them. The main focus of data flow diagrams is the processes or activities that are performed in the system. Data flow diagrams model objects, associations, and activities by describing how data flows between and around various objects, they illustrate how data is processed by a system in terms of inputs and outputs. They are pipelines through which packets of information flow. Data flow diagrams work on the premise that for every activity there is some communication, transference or flow that can be described as a data element.

Data flow diagrams describe what activities are occurring to fulfill a business relationship or accomplish a task, not how these activities are to be performed. It shows the logical sequence of associations and activities, not physical processes.

4.2.1 Context Diagram

The first DFD in every business process model, whether a manual system or a computerized system, is the context diagram. The context diagram shows the entire system in context with its environment. The context diagram shows the overall business process as just one process (i.e., the system itself) and shows the data flows to and from external entities.

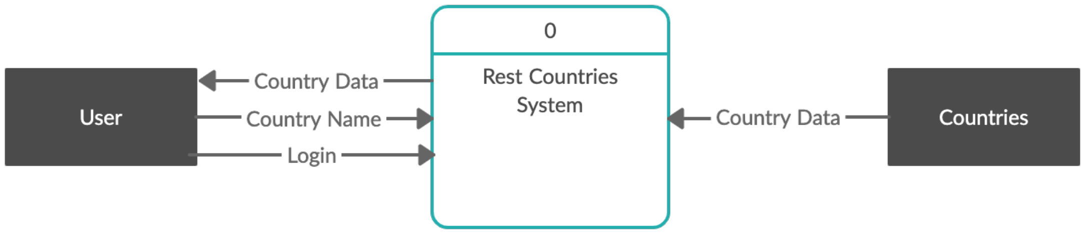


Figure 4.1: Context Flow Diagram

4.2.2 Level 1 Data Flow Diagram

Level 1 DFD is an expansion of the context diagram that shows more processes and how the users interact with the system in terms of inputs and outputs. This is the detailed description of the processes that occur in the system while related to an external entity.

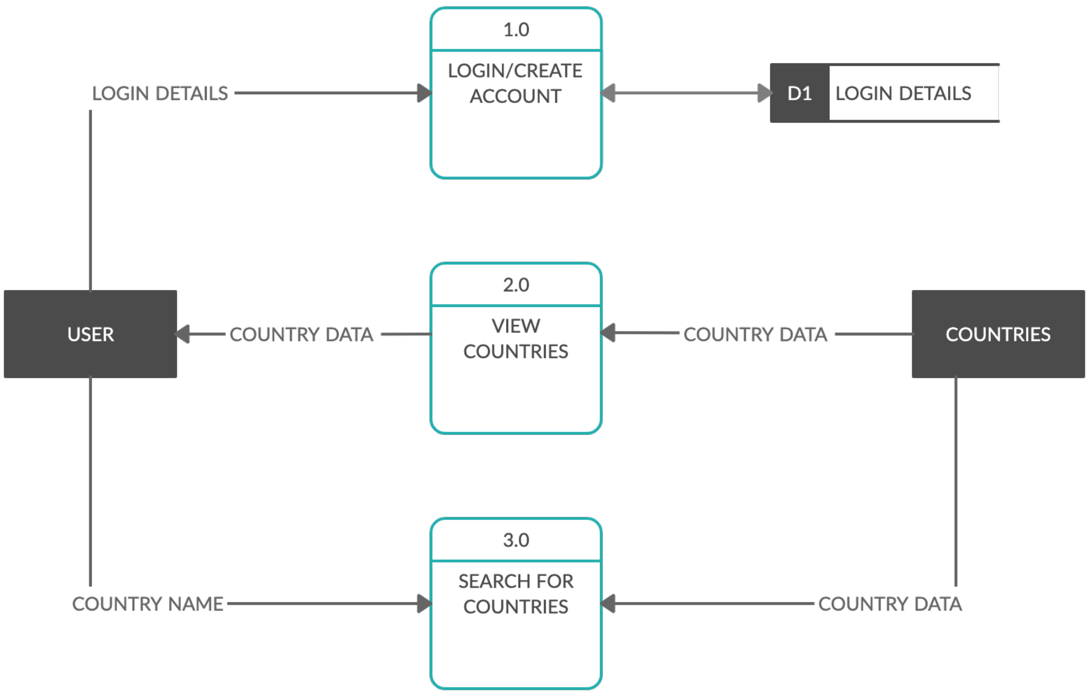


Figure 4.2: Level 1 Data Flow Diagram

4.2.3 Flow Chart

A flowchart is a type of diagram that represents a work flow or process. They are used in analysing, designing, documenting or managing a process in various fields.

Figure 4.1 shows a flowchart representing a systemic flow of how users interact with the system.

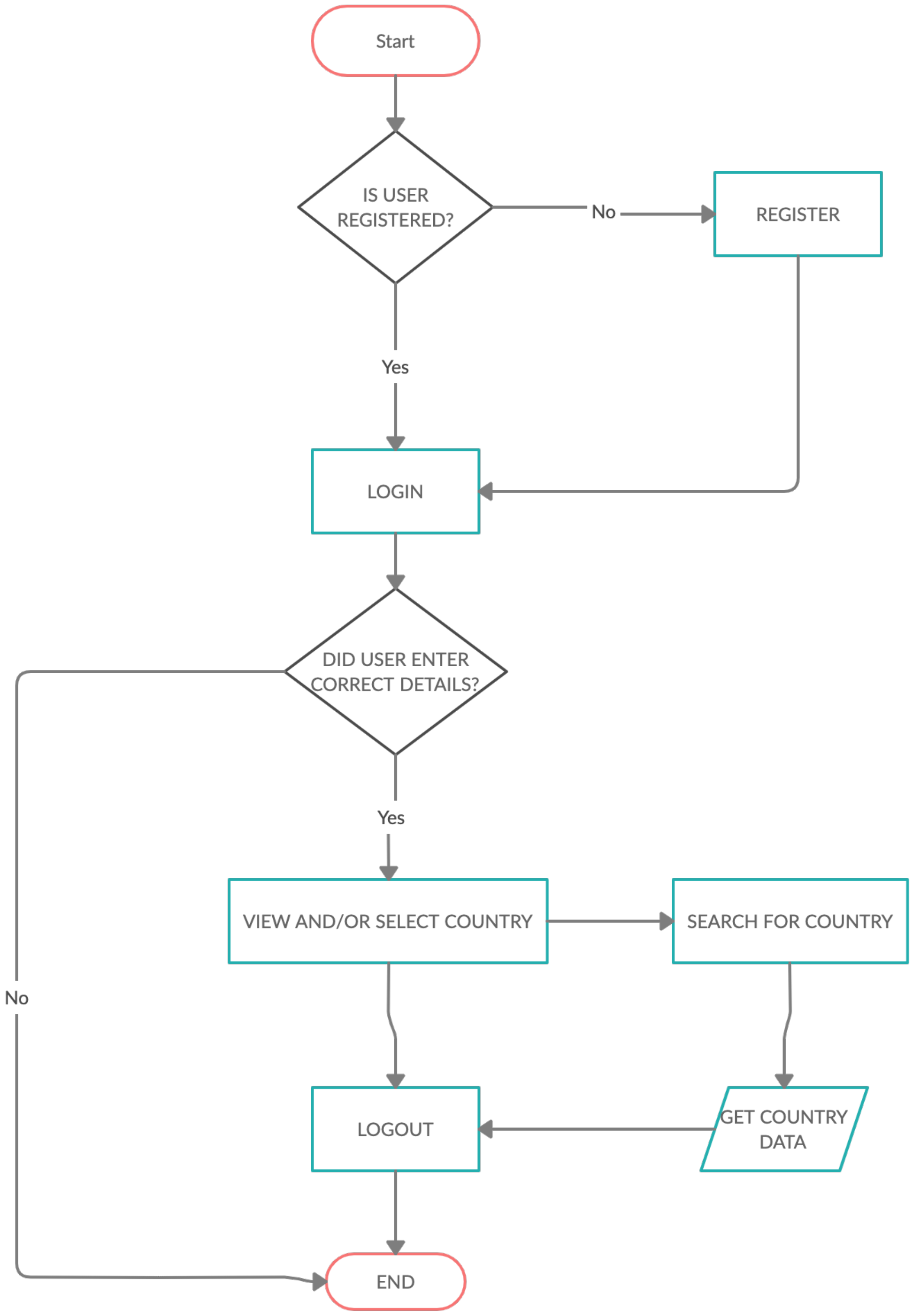


Figure 4.3: Flow Chart Diagram

4.3 Use Case

Use cases are used to explain and document the interaction that is required between the user and the system to accomplish the user's task.

Use case diagrams describe what a system does from the standpoint of an external observer. The emphasis of use case diagrams is on what a system does rather than how. They are used to show the interactions between users of the system and the system. A use case represents the several users called actors and the different ways in which they interact with the system.

Below is the use case diagram for the proposed system.

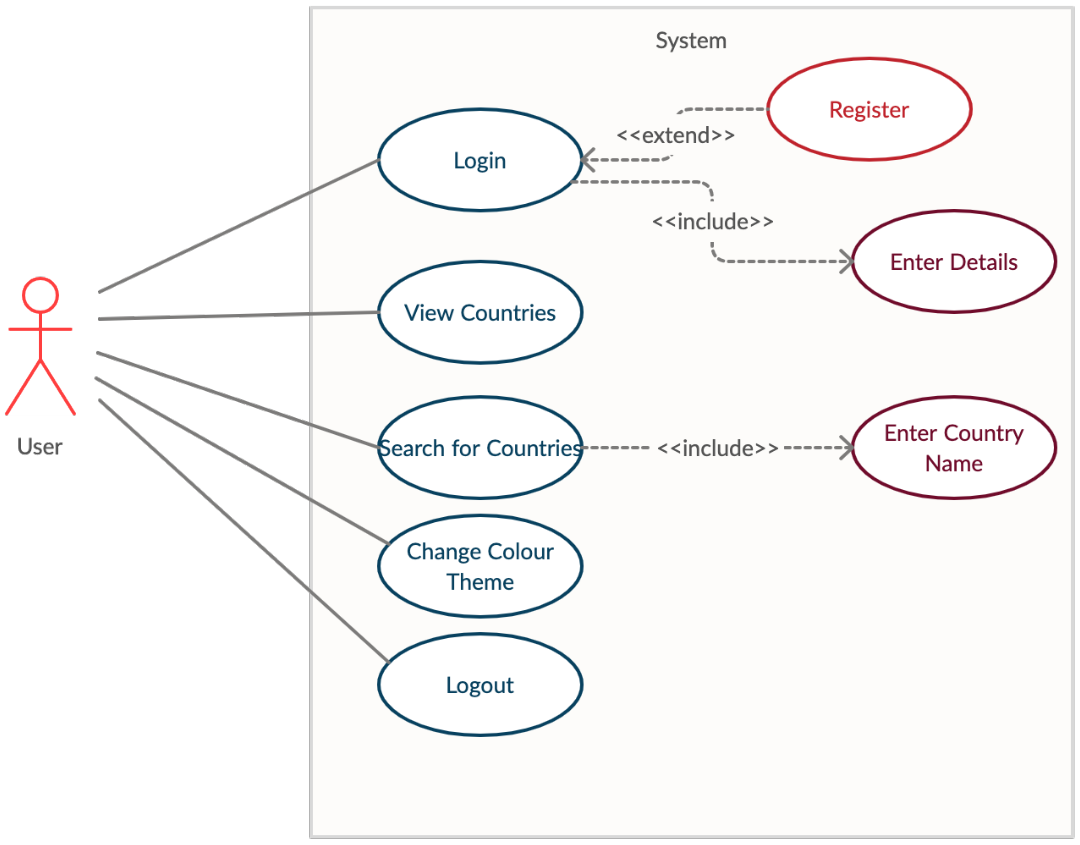


Figure 4.4: Use Case Diagram

4.4 Data Modelling and Entity Relationships

4.4.1 Data Modelling

A data model is a formal way of representing the data that are used and created by a system; it illustrates people, places, or things about which information is captured and how they are related to each other.

In system design, analysts draw a physical data model to reflect how the data will be physically be stored in the databases and files.

4.4.2 Entity Relationship Diagram (ERD)

An entity relationship diagram is a picture which shows information that is created, stored, and used by a system. On an ERD, similar kinds of information are listed together and placed inside boxes called entities. Lines are drawn between entities to represent relationships among the data, and special symbols are added to the diagram to communicate high-level business rules that need to be supported by the system.

Entities

Entities are the basic building blocks of a data model. It is a person, place, event, or thing about which data is collected.
The entities in this application include:

- Country
- Continent
- State/Province
- Currency
- Language.

Relationships

Relationships are associations between entities, and they are shown by lines that connect the entities together. Figure 4.5 below show the description of the various relationships between the entities with their cardinalities.

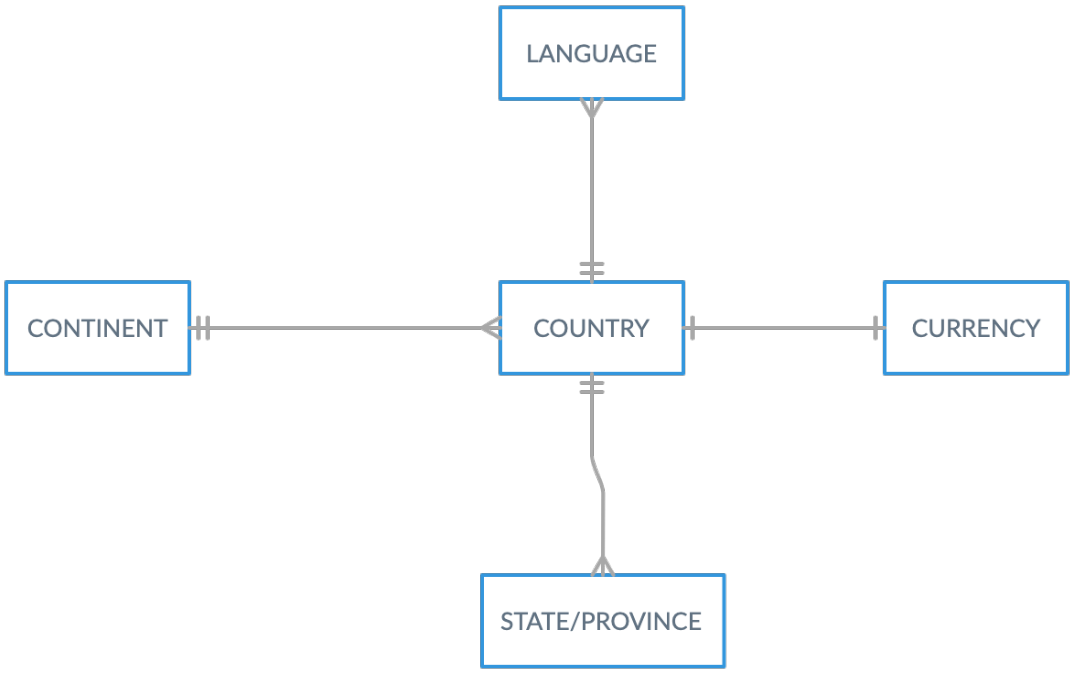


Figure 4.5: The Relationships Between the Entities

Where:

- A Continent has many countries so it represents a One-to-Many Relationship
- A Country has many States/Provinces so it represents a One-to-Many Relationship
- A Country has only one Currency so it represents a One-to-One relationship
- A Country can have one or more Languages so it represents a One-to-Many relationship

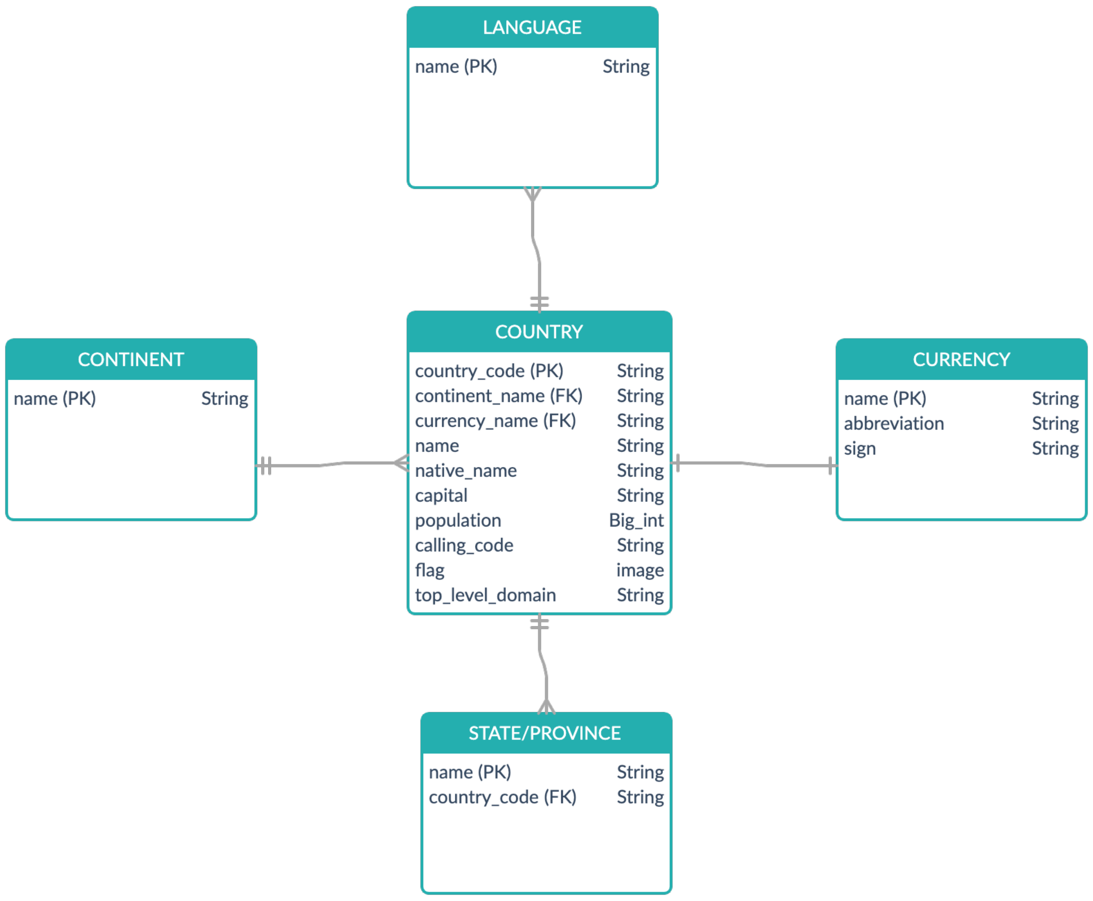


Figure 4.6: Entity Relationship Diagram

Chapter 5

Implementation

5.1 Introduction

Implementation is the stage in the project where the theoretical design is turned into a working system. System implementation is the phase where the application is realised. It involves careful planning, investigation of the current system and its constraints on implementation. The implementation process begins with preparing a plan for the implementation of the system and then proceeds to the activities to be carried out such as creating a mock-up or wireframe and then writing the code and then on to the deployment of the application.

Implementation is the most important phase. The most critical stage in achieving a successful new system is giving the users confidence that the new system will work and be effective. The system can only be implemented after thorough testing is done and it is found to be working according to the specifications.

5.2 Technologies Used to Build the System

The application was built based on a REST API called "REST Countries" which is a simple web API for getting information about the world's nations via REST calls and the API provided the information about the countries in the application.

The application was built by mainly using a popular JavaScript framework called *Vue.js*. Vue.js is an open-source front-end JavaScript framework for building interfaces and single-page applications. It features an incrementally adaptable architecture that focuses on declarative rendering and component composition.

A state manager called *Vuex* which is the primary state manager for Vue.js was used in the application to store data that was going to be used across the components for easy retrieval. Routing was also implemented in this project to enable the functionality of moving from one page to another and back and it was done mainly using *vue-router* which is the official router for Vue.js. For the styling, *SASS* which is a CSS preprocessor was used to implement the styling of the web application.

Node.js was used together with Express.js to develop the server-side of the application. Node.js which is an open-source, cross-platform, back-end, JavaScript runtime environment that executes JavaScript code outside the browser. It is a programming language that lets developers use JavaScript to write command line tools and for server-side scripting to produce dynamic web page content before the page is sent to the user's browser.

Express.js also referred to as Express, is a free and open-source back-end web application framework for Node.js. It is designed for building web applications and APIs.

MongoDB was used as the database application used to store the user information. MongoDB is a cross-platform document-oriented database program. It is classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas.

5.3 Description of the User Interface

The application has the following pages:

5.3.1 Login/Register Page

The application has been designed to ensure that access to the application can only be granted to authenticated users. The user is required to enter a valid username and password before access can be granted. Upon entering the right credentials, the user is directed to relevant sections and can now use the application (see Figure 5.1). If a user is not registered in the system then the user is required to register to

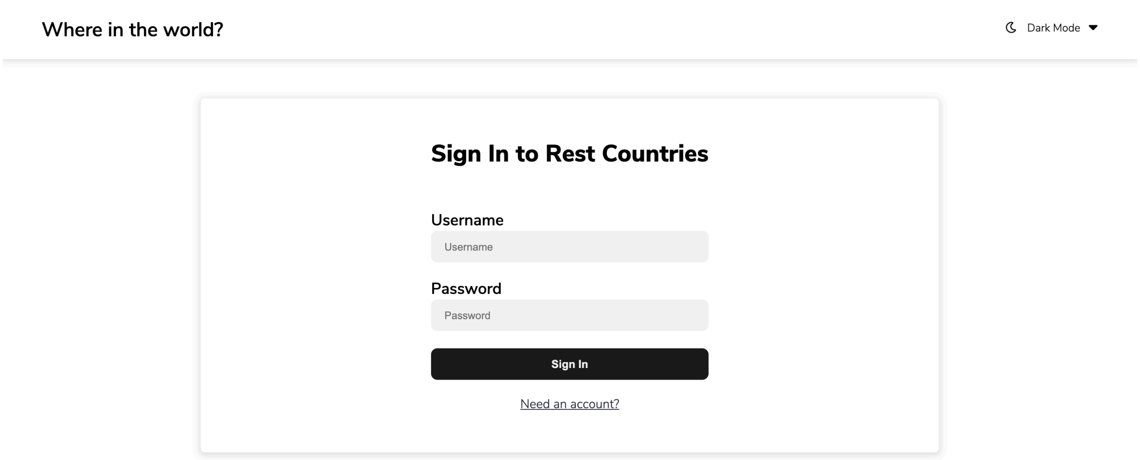


Figure 5.1: Login Page

have an account so that the user can gain access to the system (see Figure 5.2).

5.3.2 The Home Page

The home page displays all the countries in cards. It also features a search box that lets the user type in the name of any country to be searched for and the results are shown in real-time (see Figure 5.4). There is also a dropdown feature that enables the user to filter the countries by their continents such as Africa, Asia, etc. (see Figure 5.3)

5.3.3 The Descriptions Page

When a card on the home page is clicked, the user is automatically redirected to the descriptions page and this page displays the full information about the particular country that was selected (see Figure 5.5).

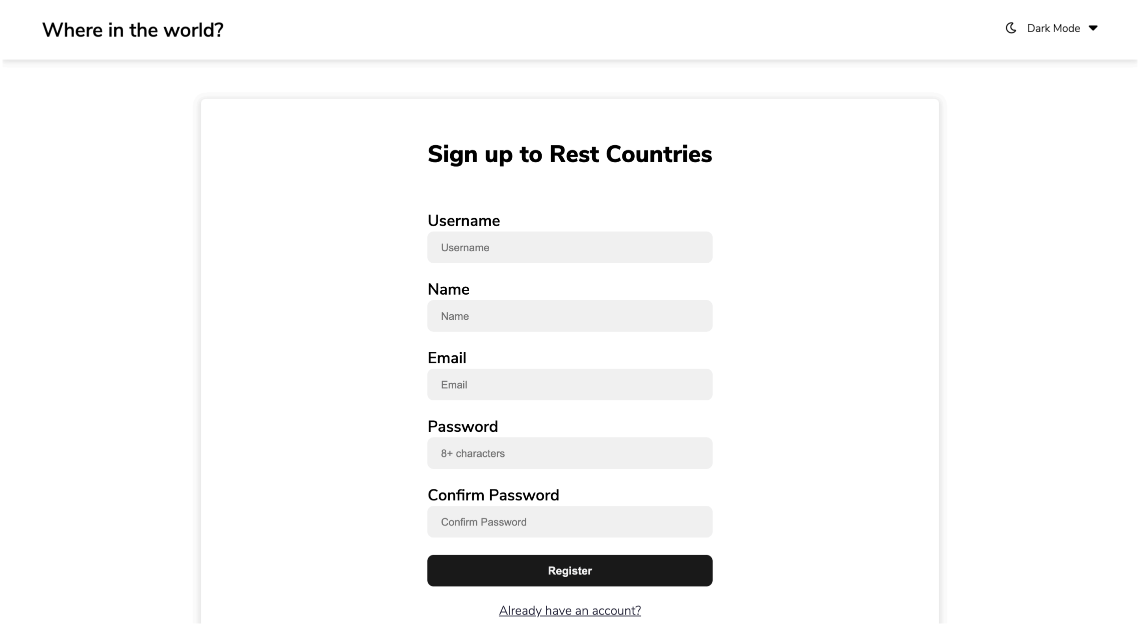


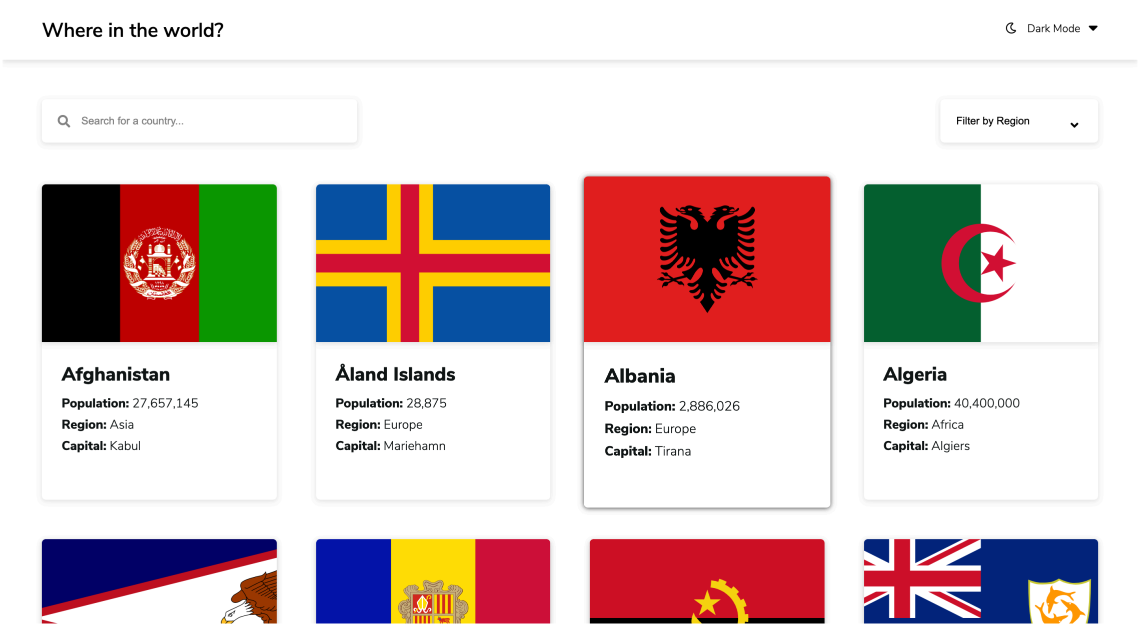
Figure 5.2: Register Page

5.3.4 Colour Switcher

The application also features a colour switcher which enables the user to toggle a colour switch to change the design theme of the application from light mode to dark mode (see Figure 5.6).

5.4 Implementing Interactivity

One of the main concerns of the implementation phase is the interface design. Therefore, it is important that the interface is designed in a way that it eases user to accomplish their task by interacting with the system efficiently and effectively. A well-designed interface possesses the ability to attract and change user's perception towards a system. In this project, the system interfaces were designed by considering various interactive design elements as to promote the better interaction of user and the system.



Where in the world?

Dark Mode

Search for a country...

Filter by Region

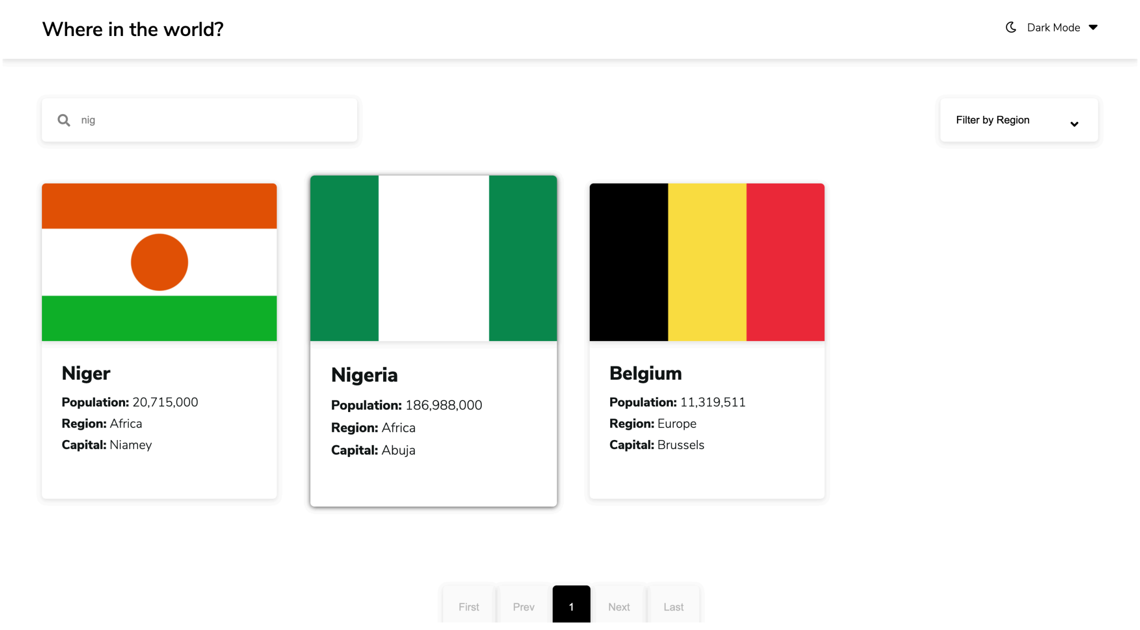
Afghanistan
Population: 27,657,145
Region: Asia
Capital: Kabul

Åland Islands
Population: 28,875
Region: Europe
Capital: Mariehamn

Albania
Population: 2,886,026
Region: Europe
Capital: Tirana

Algeria
Population: 40,400,000
Region: Africa
Capital: Algiers

Figure 5.3: Home Page



Where in the world?

Dark Mode

nig

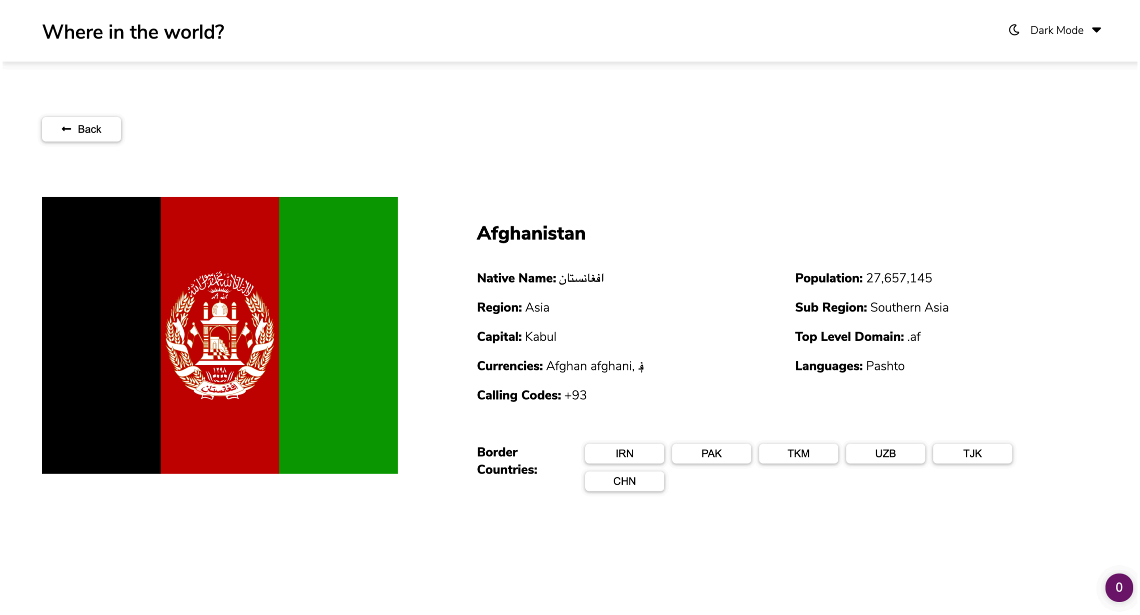
Filter by Region

Niger
Population: 20,715,000
Region: Africa
Capital: Niamey

Nigeria
Population: 186,988,000
Region: Africa
Capital: Abuja

Belgium
Population: 11,319,511
Region: Europe
Capital: Brussels

Figure 5.4: Searching for a Country



Where in the world?

Dark Mode

Back

Afghanistan

Native Name: افغانستان

Region: Asia

Capital: Kabul

Currencies: Afghan afghani, ؋

Calling Codes: +93

Population: 27,657,145

Sub Region: Southern Asia

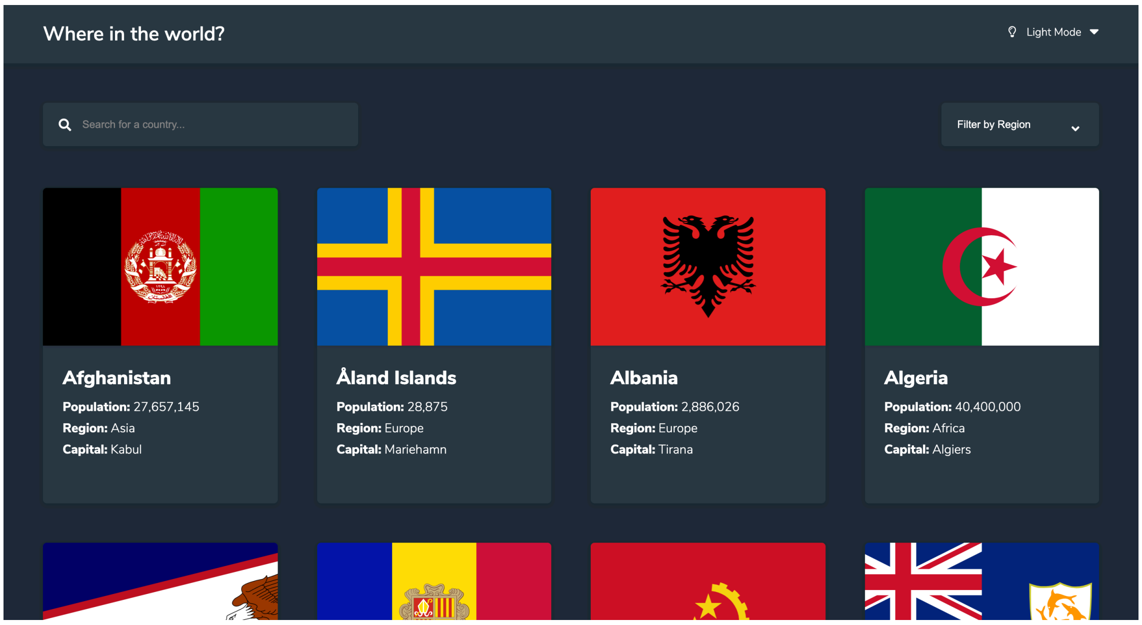
Top Level Domain: .af

Languages: Pashto

Border Countries: IRN PAK TKM UZB TJK CHN

0

Figure 5.5: Descriptions Page



Where in the world?

Light Mode

Search for a country...

Filter by Region

Afghanistan
Population: 27,657,145
Region: Asia
Capital: Kabul

Åland Islands
Population: 28,875
Region: Europe
Capital: Mariehamn

Albania
Population: 2,886,026
Region: Europe
Capital: Tirana

Algeria
Population: 40,400,000
Region: Africa
Capital: Algiers

Figure 5.6: Application in Dark Mode

Chapter 6

System Testing

6.1 Introduction

System testing is the stage of implementation which aimed at ensuring that the system works accurately and efficiently before live operation commences. Testing is the process of executing the program with the intent of finding errors and missing operations and also a complete verification to determine whether the objectives are met and the user requirements are satisfied. The ultimate aim is quality assurance.

Tests are carried out and the results are compared with the expected document. In the case of erroneous results, debugging is done.

6.2 Testing Process

The testing process is formalised by referring to three distinct goals:

- Verification
- Validation
- Evaluation

6.2.1 Verification

Verification is testing that establishes whether the system is correctly computing the required results and correctly implements the underlying theory. Simply stated, verification answers the questions: *Have I built the system right? Is it computing the right answer?*

The verification tests exercise each component independently and also the system as a whole; first checking that each individual component works correctly and then checking that they have been integrated correctly.

6.2.2 Validation

Validation is the testing that establishes whether the system meets the user's requirements. In this case, the questions are: *Have I built the right system? Does it satisfy the requirements?*. Comparing the system's behaviour with the original requirements and system specifications;

- The system is built as a web application so it can be accessed on any device
- The system design is simple and interactive
- The system does show information about any country in the world.

So in essence, it can be said that the system satisfies the major requirements.

6.2.3 Evaluation

In evaluation tests, we ask: *How good is the system?* A problem encountered when building the system was how to get information about the countries, getting information personally about all the countries would have been too time tasking seeing as the time frame for working on the project was small. Therefore, an API was used and the system of using an API is What You See Is What You Get (WYSIWYG). The API used only provided little information about the countries and did not go into detail about the information. Nevertheless, the API served to good use as it still provided some very important information about the countries that the users of the system can work with.

6.3 Forms of Testing

There are four main forms of testing which were employed in the building of this application:

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

6.3.1 Unit Testing

The software units in the system are modules and routines assembled and integrated to perform a specific function. This application was built by dividing each functionality into components so it was easy to perform unit tests on the application as the application was already in components.

As a part of the unit testing, the program was executed for the individual components independently. This enables easy detection of errors in the coding and logic that are contained in each of the components.

6.3.2 Integration Testing

Data can be lost across any interface, one module can have adverse effects on another, sub-functions when combined may not produce the desired major functions.

Integration testing is a systematic testing to discover errors associated within the interface. They ensure that the interfaces and linkages between different parts of the system work properly. The objective is to take unit tested components and build a program structure. All the components are combined and tested as a whole. This testing provides the assurance that the application is a well integrated functional unit with smooth transition of data.

6.3.3 System Testing

System testing is the key factor for the success of any system. It ensures that all the components and programs work together without error. System testing is similar to integration testing, but is much broader in scope. Whereas integration testing focuses on whether the modules work together without error, system tests examine how well the system meets business requirements and its usability and security.

Security Testing

This was a process that was intended to reveal flaws in the security mechanisms of the system to protect data and maintain functionality as intended. For example, authentication of the system could only allow registered users access to the system information.

In case the user tries to log into the system using a wrong user credentials, the system shows an error message telling the user that you have input a wrong username or password and should try again or create a new account.

6.3.4 Acceptance Testing

Acceptance tests are done primarily by the users. The goal is to confirm that the system is complete, meets the user requirements that prompted the system to be developed, and is acceptable to the users.

Chapter 7

Conclusion

7.1 Introduction

Through the system analysis and design, and the implementation of the designed system, we were able to reflect the requirements of the objectives of the study. This chapter gives a summary of what was done in all the chapters, outcomes, limitations, and lessons learnt.

During the development of the Rest Countries Application, a number of activities were carried out and these included; System analysis which specified the major requirements for the system and the best technology to use to develop the system. In the system design, the entity relationship diagrams and data flow diagrams were used to identify entities, relationships, attributes in the system.

The application was built, and I was exposed to practical challenges, and enhanced application of knowledge in solving real world problems.

7.2 Limitations

One of the greatest limitations of the study was the time frame required for the project which was short and then the lack of detailed information from the API about the countries also served as a limitation. Also, unavailability of literature and other reading materials relating to our system also slowed down the progress of the report since there are few or limited resource of such information.

7.3 Final Conclusion

In this study, we evaluated the existing different activities that are conducted during the finding, the system was analysed, designed, implemented, and tested. The result was a working interactive system that enables users to search for countries and view information about those countries.

Appendix A

Source Code

A.1 Importing the Servers

```
const express = require("express");
const mongoose = require("mongoose");
const bodyParser = require("body-parser");
const path = require("path");
const cors = require("cors");
const passport = require("passport");

/** Initialise the application and setting up Express */
const app = express();

/** Define some Middleware */

/** Form Data Middleware */
app.use(
  bodyParser.urlencoded({
    extended: false,
  })
);
```

```
/** JSON Body Middleware */
app.use(bodyParser.json());

/** CORS Middleware */
app.use(cors());

// Apply Path Modules
// Setting up the static directory
app.use(express.static(path.join(__dirname, "public")));

// Use the passport Middleware
app.use(passport.initialize());

// Bring in the passport Strategy
require("./config/passport")(passport);

// Bring in the Database Config and connect with the
database
const db = require("./config/keys").mongoURI;
mongoose
  .connect(db, { useNewUrlParser: true })
  .then(() => {
    console.log(`Database connected successfully ${db}`);
  })
  .catch((err) => console.log(`Unable to connect with the
  database ${err}`));

// Bring in the Users route
const users = require("./routes/api/users");
app.use("/api/users", users);

app.get("*", (req, res) => {
  res.sendFile(path.join(__dirname, "public/index.html"));
});

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
  console.log(`Server started on port ${PORT}`);
});
```

A.2 Specifying User Attributes

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

// Create the User Schema
const UserSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    username: {
        type: String,
        required: true
    },
    email: {
        type: String,
        required: true
    },
    password: {
        type: String,
        required: true
    },
});
```

```
module.exports = User = mongoose.model('users',UserSchema);
```

A.3 Creating the Functions for Login/Registration

```
const express = require("express");
const router = express.Router();
/** In order to hash our passwords */
const bcrypt = require("bcryptjs");
/** Also used to hash our passwords too */
const jwt = require("jsonwebtoken");
const passport = require("passport");
const key = require("../../config/keys").secret;
const User = require("../../model/User");
const emailRegex = /^(([^<>()\[\]\\.,;:\s@"]+(\.[^<>()\
[\]\\.,;:\s@"]+)*)|(".+"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.
[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]
{2,}))$/;

/**
 * @route POST api/users/register
 * @desc Register the user
 * @access Public
*/
router.post("/register", (req, res) => {
  let { name, username, email, password,
  confirm_password } = req.body;

  if (password !== confirm_password) {
    return res.status(400).json({
      msg: "Password does not match",
    });
  } else if (password.length < 8) {
    return res.status(400).json({
      msg: "Password should be more than eight characters",
    });
  }

  // Check for the unique username
  User.findOne({ username: username }).then((user) => {
    if (user) {
      return res.status(400).json({
        msg: "Sorry, that username is already taken.",
      });
    }
  });

  // Check for the unique Email
  User.findOne({ email: email }).then((user) => {
    if (user) {
      return res.status(400).json({
        msg: "email is already registered.
        Did you forget your password?",
      });
    }
  });

  // Validate email
  const valid = emailRegex.test(email);
  //   const parts = email.split("@");
  //   const domainParts = parts[1].split(".");

  if (email.length > 254) {
    return res.status(400).json({
      msg: "Please enter a valid email address",
    });
  } else if (!valid) {
    return res.status(400).json({
      msg: "Please enter a valid email address",
    });
  }

  // The data is valid and now we can register the user
  let newUser = new User({
    name,
    username,
    password,
    email,
  });

  // Hash the password
  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(newUser.password, salt, (err, hash) => {
      if (err) throw err;
      newUser.password = hash;
      newUser.save().then((user) => {
        return res.status(201).json({
          success: true,
          msg: "User is now registered.",
        });
      });
    });
  });
});

/**
 * @route POST api/users/login
 * @desc Signing in the User
 * @access Public
 */
router.post("/login", (req, res) => {
  User.findOne({
    username: req.body.username,
  }).then((user) => {
    if (!user) {
      return res.status(404).json({
        msg: "Username is not found.",
        success: false,
      });
    }

    // If there is user we are now going to compare
    the password
    bcrypt.compare(req.body.password, user.password)
    .then((isMatch) => {
      if (isMatch) {
        // User's password is correct and we need
        to send the JSON token for that user
        const payload = {
          _id: user._id,
          username: user.username,
          name: user.name,
          email: user.email,
        };
        jwt.sign(
          payload,
          key,
          {
            expiresIn: 604800,
          },
          (err, token) => {
            res.status(200).json({
              success: true,
              token: `Bearer ${token}`,
            //   user: user,
              msg: "You are now logged in.",
            });
          }
        );
      } else {
        return res.status(404).json({
          msg: "Incorrect password.",
          success: false,
        });
      }
    });
  });
});

/**
 * @route POST api/users/profile
 * @desc Return the User's Data
 * @access Private
 */
router.get(
  "/home",
  passport.authenticate("jwt", { session: false }),
  (req, res) => {
    return res.json({
      user: req.user,
    });
  }
);
```

```
module.exports = router;
```

A.4 The Store

```
import Vue from "vue";
import Vuex from "vuex";
import axios from "axios";
import router from "../router/index";

Vue.use(Vuex);

export default new Vuex.Store({
  state: {
    token: localStorage.getItem("token") || "",

    /** This will store the user data */
    user: {},
    status: "",
    darkMode: false,
    countries: [],
    country: [],
    error: null
  },
  /** This will change the state of the data */
  mutations: {
        changeColor: (state, darkMode) =>
        (state.darkMode = darkMode),
        countryData: (state, countries) =>
        (state.countries = countries),
    country: (state, country) => (state.country = country),
    auth_request: state => {
      state.status = "loading";
      state.error = null;
    },
    auth_success: (state, token, user) => {
      state.token = token;
      state.user = user;
      state.status = "success";
      state.error = null;
    },
    auth_error(state, err) {
      state.error = err.response.data.msg;
    },
    register_request: state => {
      state.status = "loading";
      state.error = null;
    },
    register_success: state => {
      state.status = "success";
      state.error = null;
    },
    register_error(state, err) {
      state.error = err.response.data.msg;
    },
    logout: state => {
      state.status = "";
      state.token = "";
      state.user = "";
      state.error = null;
    }
  },
  actions: {
    toggleColor({ commit }) {
      this.darkMode = !this.darkMode;
      commit("changeColor", this.darkMode);
  },
  async fetchCountryData({ commit }) {
    const response = await axios.get(
      `https://restcountries.eu/rest/v2/all?fields=flag;
      name;population;region;capital;alpha3Code;`
    );
    commit("countryData", response.data);
  },
  async fetchRegions({ commit }, region) {
    const response = await axios.get(
      `https://restcountries.eu/rest/v2/region/${region}
      ?fields=flag;name;population;region;capital;`
    );
    commit("countryData", response.data);
  },
  async searchCountry({ commit }, input) {
    const response = await axios.get(
      `https://restcountries.eu/rest/v2/name/${input}`
    );
    commit("countryData", response.data);
  },
  async searchFullCountry({ commit }, input) {
    const response = await axios.get(
      `https://restcountries.eu/rest/v2/name/${input}
      ?fullText=true`
    );
    commit("country", response.data);
  },

  // Login Action
  async login({ commit }, user) {
    commit("auth_request");
    try {
      let res = await axios.post("/api/users/login",user);
      if (res.data.success) {
        const token = res.data.token;
        const user = res.data.user;
        // Store the token into the localStorage
        localStorage.setItem("token", token);
        // Set the axios defaults
      axios.defaults.headers.
      common["Proxy-Authorization"] = token;
      commit("auth_success", token, user);
      }
      return res;
    } catch (err) {
      commit("auth_error", err);
    }
  },

  // Register User
  async register({ commit }, userData) {
    commit("register_request");

    // axios.post() since we're sending data
    try {
      let res = await axios.post("/api/users/register",
      userData);
      if (res.data.success !== undefined) {
        commit("register_success");
      }
      return res;
    } catch (err) {
      commit("register_error", err);
    }
  },
    // Logout function
    async logout({ commit }) {
      await localStorage.removeItem("token");
      commit("logout");
        delete axios.defaults.headers
        .common["Proxy-Authorization"];
      router.push("/login");
      return;
    }
  },
  getters: {
    allCountries: state => state.countries,
    isLoggedIn: state => !!state.token,
    authState: state => state.status,
    user: state => state.user,
    error: state => state.error
  },
  modules: {}
});
```

A.5 Creating the Routes

```
import Vue from "vue";
import VueRouter from "vue-router";
import Home from "../views/Home.vue";
import store from "../store/index";

Vue.use(VueRouter);

const routes = [
  {
    path: "/",
    name: "Home",
    component: Home,
    meta: {
      authRequired: true
    }
  },
  {
    path: "/login",
    name: "Login",
    component: () => import("../views/Login.vue"),
    meta: {
      requiresGuest: true
    }
  },
  {
    path: "/register",
    name: "Register",
    component: () => import("../views/Register.vue"),
    meta: {
      requiresGuest: true
    }
  },
  {
    path: "/description",
    name: "Description",
    component: () => import("../views/Description.vue"),
    meta: {
      authRequired: true
    }
  }
];
```

```
const router = new VueRouter({
  mode: "history",
  base: process.env.BASE_URL,
  routes
});

router.beforeEach((to, from, next) => {
  if (to.matched.some(record => record.meta.authRequired))
  {
    if (!store.getters.isLoggedIn) {
      // Redirect to the Login Page
      next("/login");
    } else {
      next();
    }
  } else if (to.matched.some(record => record.meta
  .requiresGuest)) {
    if (store.getters.isLoggedIn) {
      // Redirect to the Home Page
      next("/");
    } else {
      next();
    }
  } else {
    next();
  }
});
export default router;
```

A.6 Register Component

```
<template>
  <div :class="{ body__dark: darkMode }">
    <Heading />
    <div class="form" :class="{ form__dark: darkMode }">
      <h2 class="form__heading">
        Sign up to Rest Countries
      </h2>
        <form class="form__body" @submit.prevent=
        "registerUser">
        <div class="form__group">
          <label class="form__label" for="username">
           Username
          </label>
          <input
            id="username"
            type="text"
            placeholder="Username"
            name="username"
            v-model="username"
            class="form__control"
            :class="{ form__control__dark: darkMode,
            error: includesUser() }"
          />
          <small v-if="error" class="small__text">{{
            error.includes("username") ? error : ""
          }}</small>
        </div>

        <div class="form__group">
        <label class="form__label" for="name">
          Name
        </label>
          <input
            id="name"
            type="text"
            placeholder="Name"
            name="name"
            v-model="name"
            class="form__control"
            :class="{ form__control__dark: darkMode }"
          />
        </div>

        <div class="form__group">
        <label class="form__label" for="email">
            Email
        </label>
          <input
            id="email"
            type="text"
            placeholder="Email"
            name="email"
            v-model="email"
            class="form__control"
            :class="{ form__control__dark: darkMode,
            error: includesMail() }"
          />
          <small v-if="error" class="small__text">{{
            error.includes("email") ? error : ""
          }}</small>
        </div>

        <div class="form__group">
        <label class="form__label" for="password">
            Password
        </label>
          <input
            id="password"
            type="password"
            placeholder="8+ characters"
            name="password"
            v-model="password"
            class="form__control"
            :class="{ form__control__dark: darkMode,
            error: includesPass() }"
          />
          <small v-if="error" class="small__text">{{
            error.includes("Password") ? error : ""
          }}</small>
        </div>

        <div class="form__group">
          <label class="form__label" for="confirm_password"
            >Confirm Password</label
          >
          <input
            id="confirm_password"
            type="password"
            placeholder="Confirm Password"
            name="password"
            v-model="confirm_password"
            class="form__control"
            :class="{ form__control__dark: darkMode,
            error: includesPass() }"
          />
          <small v-if="error" class="small__text">{{
            error.includes("Password") ? error : ""
          }}</small>
        </div>
        <button class="form__btn" :class=
        "{ form__btn__dark: darkMode }">
          Register
        </button>
        <p class="form__link" :class=
        "{ form__link__dark: darkMode }">
          <router-link
            to="/login"
            class="form__link"
            :class="{ form__link__dark: darkMode }"
            >Already have an account?</router-link
          >
        </p>
      </form>
    </div>
  </div>
</template>

<script>
import Heading from "../components/Heading";
import { mapState, mapActions, mapGetters } from "vuex";

export default {
  name: "Register",
  data() {
    return {
      username: "",
      password: "",
      confirm_password: "",
      name: "",
      email: "",
      pass: "Password",
      user: "username",
      mail: "email"
    };
  },
  components: {
    Heading
  },
  computed: {
    ...mapState(["darkMode"]),
    ...mapGetters(["error"])
  },
  methods: {
    ...mapActions(["register"]),
    includesPass() {
      if (this.error) {
        return this.error.includes(this.pass);
      }
    },
    includesUser() {
      if (this.error) {
        return this.error.includes(this.user);
      }
    },
    includesMail() {
      if (this.error) {
        return this.error.includes(this.mail);
      }
    },
    registerUser() {
      let user = {
        username: this.username,
        password: this.password,
        confirm_password: this.confirm_password,
        email: this.email,
        name: this.name
      };
      this.register(user).then(res => {
        if (res.data.success) {
          this.$router.push("/");
        }
      });
    }
  }
};
</script>

<style></style>
```

A.7 Login Component

```
<template>
  <div :class="{ body__dark: darkMode }">
    <Heading />
    <div class="form" :class="{ form__dark: darkMode }">
      <h2 class="form__heading">
        Sign In to Rest Countries
      </h2>
      <form class="form__body" @submit.prevent="loginUser">
        <div class="form__group">
        <label class="form__label" for="username">
                Username
        </label>
          <input
            id="username"
    type=”text”
    placeholder=”Username”
    name=”username”
    v–model=”username”
    class=”form__control”
    :class=”{ form__control__dark: darkMode,
    error: includesUser() }”
  />
  <small v–if=”error” class=”small__text”>{{
    error.includes(user) ? error : ””
  }}</small>
</div>

<div class=”form__group”>
<label class=”form__label” for=”password”>
        Password
</label>
  <input
    id=”password”
    type=”password”
    placeholder=”Password”
    name=”password”
    v–model=”password”
    class=”form__control”
    :class=”{
      form__control__dark: darkMode,
      error: includesPass()
    }”
  />
  <small v–if=”error” class=”small__text”>{{
    error.includes(pass) ? error : ””
  }}</small>
</div>
```

```
        <button class="form__btn"
        :class="{ form__btn__dark: darkMode }">
          Sign In
        </button>
        <p class="form__link"
        :class="{ form__link__dark: darkMode }">
          <router-link
            to="/register"
            class="form__link"
            :class="{ form__link__dark: darkMode }"
            >Need an account?</router-link
          >
        </p>
      </form>
    </div>
  </div>
</template>

<script>
import Heading from "../components/Heading";
// import store from "../store/index"
import { mapState, mapActions, mapGetters } from "vuex";

export default {
  name: "Register",
  data() {
    return {
      username: "",
      password: "",
      pass: "password",
      user: "Username"
    };
```

```
},
components: {
  Heading
},
computed: {
  ...mapState(["darkMode"]),
  ...mapGetters(["error"])
},
methods: {
  ...mapActions(["login"]),
  includesPass() {
    if (this.error) {
      return this.error.includes(this.pass);
    }
  },
  includesUser() {
    if (this.error) {
      return this.error.includes(this.user);
    }
  },
  loginUser() {
    let user = {
      username: this.username,
      password: this.password
    };
    this.login(user)
      .then(res => {
        if (res.data.success) {
          this.$router.push("/");
        }
      })
      .catch(err => {
        console.log(err);
```

```
                });
            }
        }
};
</script>

<style></style>
```

Bibliography

Adenowo, A. A., & Adenowo, B. A. (2013). Software engineering methodologies: A review of the waterfall model and object-oriented approach. *International Journal of Scientific & Engineering Research*, *4*(7), 427–434.

Aoki, K. (2000). Taxonomy of interactivity on the web. *Esitelmä Association of Internet Researchers-konferenssissa. University of Kansas. Lawrence, Kansas.(Saatavilla: http://citeseerx.ist.psu.edu/viewdoc/download.*

Dennis, A., Wixom, B. H., & Tegarden, D. (2009). *Systems analysis and design uml version 2.0*. Wiley.

Kamari, A. (2011). Interactive hostel management system.

Petersen, K., Wohlin, C., & Baca, D. (2009). The waterfall model in large-scale development. *International Conference on Product-Focused Software Process Improvement*, 386–400.