SOLUTION APPROACH DOCUMENT

DYNAMIC LOAD COMPONENT

By

CHENNA GEETHA MADHURI (VTU17345) - Testing

PRANEETH KASANAGOTTU (VTU17914) - Code & Implementation

VETUKURI NIROSHA (VTU12425) - Design and Diagrams

PAILLA BINDU SRI (VTU17363) - Design and Analysis

SHAIK ARIFULLA (VTU18208) - Report and Bug Analysis

Under the guidance of

Dr.A.Aalan Babu (Assistant Professor)

Ms.K.Nithya (Assistant Professor)

Dr.A.Udayakumar (Assistant Professor)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
SCHOOL OF COMPUTING

VEL TECH RANGARAJAN Dr. SAGUNTHALA R&D INSTITUTE OF SCIENCE AND TECHNOLOGY

ABSTRACT

When a user opens a webpage, numerous adverts are loaded at once, which makes the user less inclined to return to that page. Therefore, by utilizing the concept of a dynamic load component, the component can be loaded whenever it is needed. By enabling users to switch between various webpage elements, dynamic loading can enhance their user experience. Frontend is rendered by EJS template engine and we use NODE JS, EXPRESS JS as the backend. Our component loads the Ads in a specified location as per the Template defined by the website developer. All the data collections are stored in the MongoDB ,the image collections are specified with Start date and End date .If Start date matches with the current Date then the Ad will be displayed on the Web page .The Webpage displays the obtained ad data to the User.

KEYWORD: Dynamic Loading, MongoDB, EXPRESS JS, EJS Template engine.

LIST OF FIGURES

ABBREVIATION

HTML	Hyper Text Markup Language
API	Application Programming Interface
CSS	Cascading Style Sheet
NPM	Node Package Manager
EJS	Embedded Java Script

TABLE OF CONTENT

CHAPTER 1

INTRODUCTION

1.1 ABOUT THIS DOCUMENT

1.1.1 PURPOSE & SCOPE OF THE DOCUMENT

- The purpose of this project is to develop a web application that enables users to interact with dynamic content and advertisements.
- The scope includes components such as the Webpage, Main Component, Content Component, and MongoDB, which work together to facilitate content retrieval, ad display, and seamless communication in a way that benefits both the users and also the advertisers.
- The project aims to create an intuitive and engaging user experience, ultimately enhancing user browsing by effectively delivering relevant content and the advertisements.

1.1.2 OVERVIEW

This project aims to create a web application that provides users with a dynamic and interactive browsing experience. The project involves several components, including the Webpage, Content Component, Main Component, and Mongo DB.The Web Page sends requests to the Content Component for content, which is retrieved and delivered back to the Webpage. Additionally, the Web Page requests ads from the Main Component, which checks for ad data availability. If the collections Start time matches with the Current system date and time the Ads will be retrieved to the Webpage. Finally, the Webpage displays the ads to the user. By implementing this process flow, the project aims to enhance user engagement and satisfaction by efficiently delivering relevant content and advertisements.

CHAPTER 2

COMPONENT DESIGN

2.1 COMPONENT DESIGN DIAGRAM

In this section we describe the component design and the specific way of designing it.

2.1.1 OVERALL WORKFLOW

The component design diagram describes how the Main component and Content Component is loaded into a website.

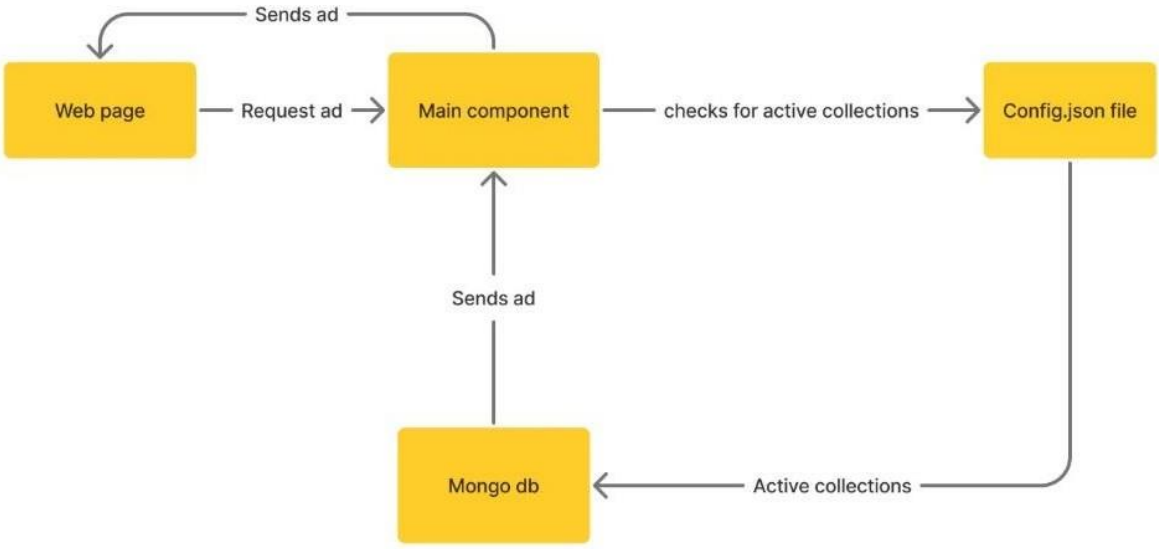

Fig 2.1.1: Overall Workflow

2.1.2 SEQUENCE DIAGRAM

The sequence diagram showcases the flow of interactions in the project. Initially, the webpage requests for the Ad component and the Content Component . The Ads are loaded as per current date given to the collections the content Component renders its heading and paragraph content. As the application runs, user interactions are enabled, allowing them to interact with the displayed components. These periodic updates provide dynamic content for the user. User interactions trigger appropriate actions and event handlers within the components, enabling a responsive and interactive experience.

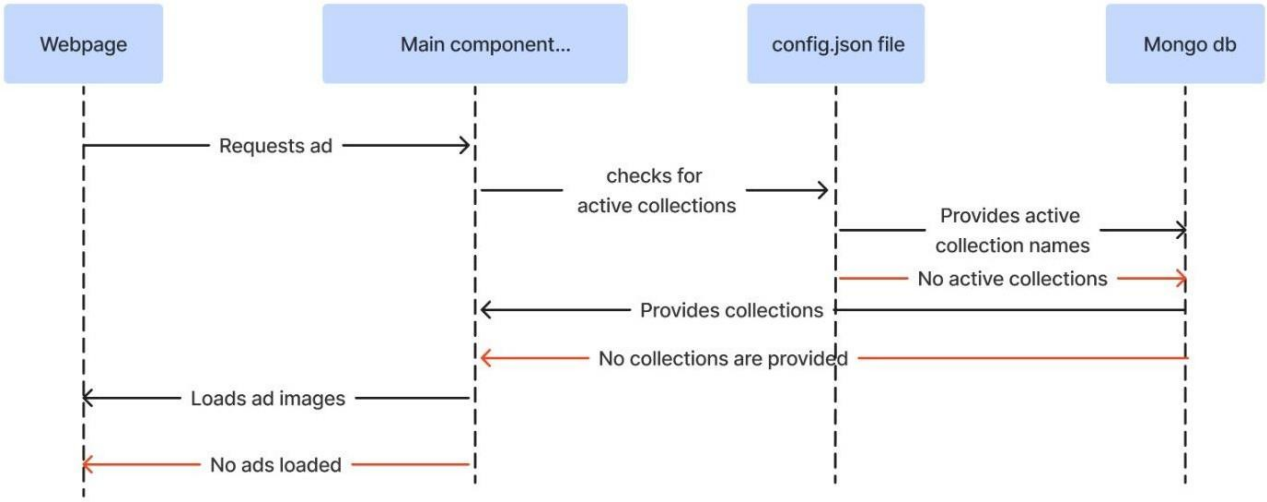

Fig 2.1.2: Sequence Diagram

2.1.3 LOW-LEVEL DESIGN:

The below low-level design diagram showcases an Express.js component that handles image collections based on the current date and a configuration file. It routes HTTP requests, retrieves the relevant collection using the Image Router, and fetches start and end dates from the Config Router. The Image Collection and Config Store services interact with the data source, allowing the component to dynamically display the appropriate collection by comparing the current date with the configured dates from the config.json file.

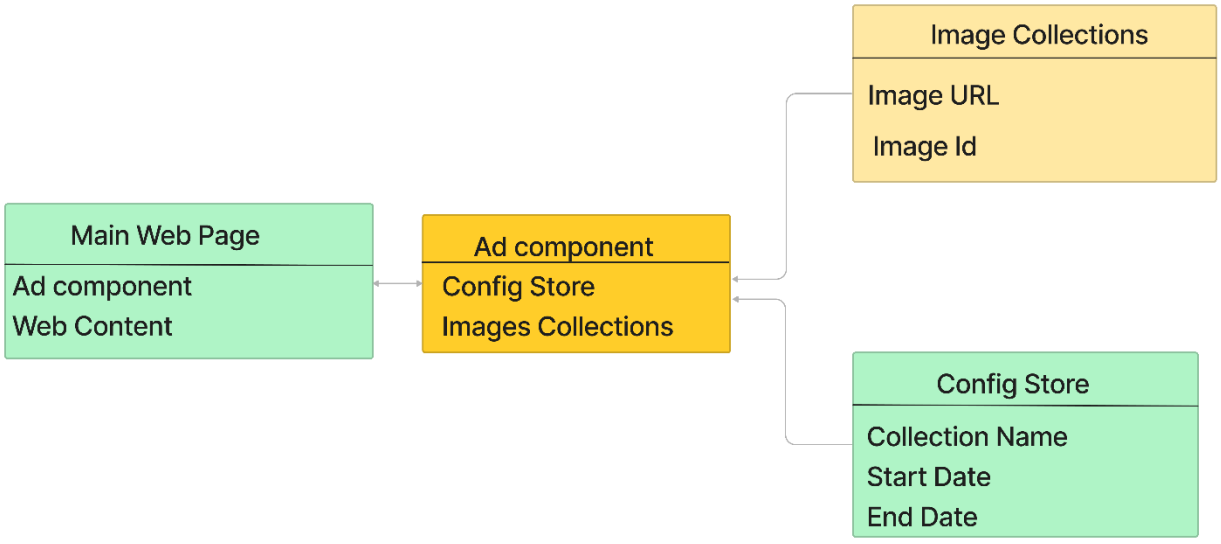

Fig 2.1.3: Low-Level Design

2.1.4 ACTIVITY DIAGRAM

The activity diagram shows how the activation process starts and ends in the web application. Activities of each module are mentioned.

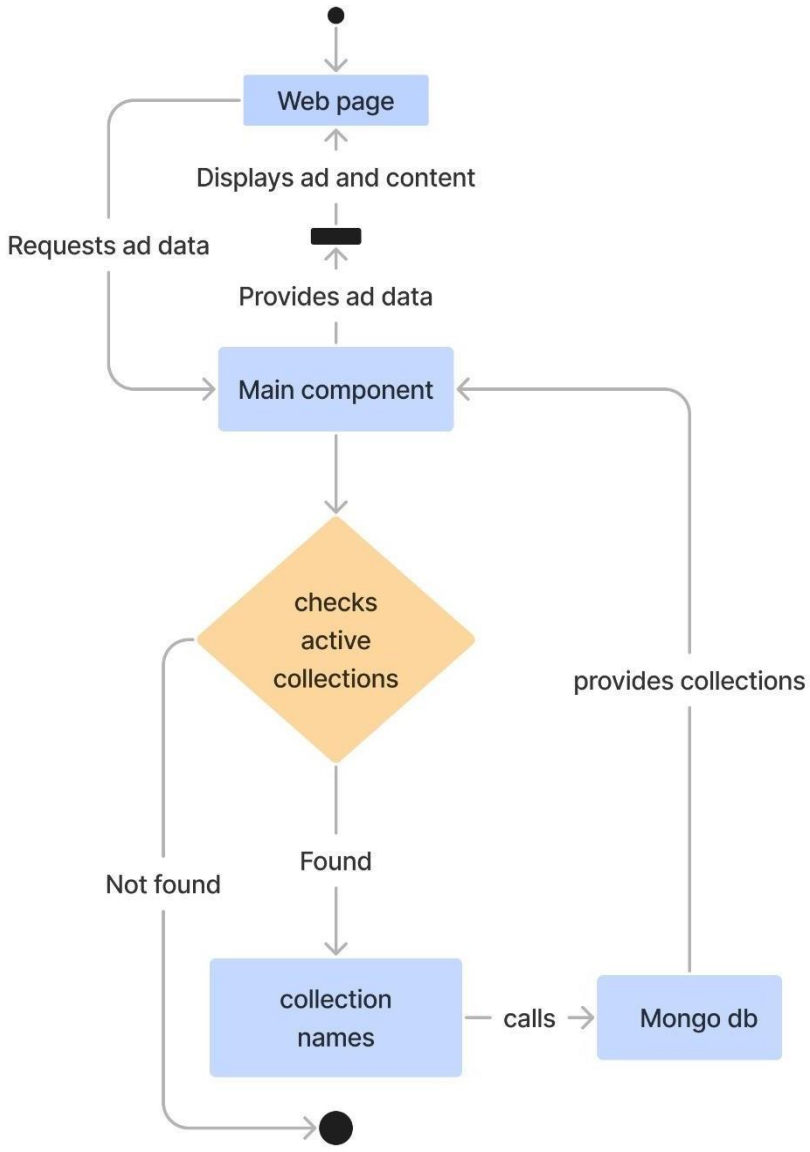

Fig 2.1.4: Activity Diagram

CHAPTER 3

TECHNOLOGY & FRAMEWORK

3.1 FRONT-END

3.1.1 EJS

EJS or Embedded Java script Templating is a templating engine used by Node.js. Template engine helps to create an HTML template with minimal code. Also, it can inject data into an HTML template on the client side and produce the final HTML. EJS is a simple templating language that is used to generate HTML markup with plain JavaScript

Purpose

EJS allows you to insert dynamic data into your HTML templates. You can use JavaScript code within your templates to generate content based on variables, perform calculations, conditionally display elements, and iterate over arrays or objects. You can define separate EJS files for different sections of your website and include them in multiple pages, making it easier to maintain and update your code.

Prerequisite

HTML,CSS &JS

HTML and JavaScript knowledge: EJS is an embedded JavaScript templating language, so it assumes that you have a solid understanding of HTML and JavaScript. Familiarity with HTML syntax, elements, and attributes is necessary to structure your web pages, while JavaScript knowledge is essential for embedding .

3.2 BACK-END

3.2.1 EXPRESS JS

Express.js is a small framework that works on top of Node.js web server functionality to simplify its APIs and add helpful new features. It makes it easier to organize your application's functionality with middleware and routing. It adds helpful utilities to Node.js HTTP objects and facilitates the rendering of dynamic HTTP objects

Purpose

Express.js is a well-known Node.js web application framework made to make it easier to create server-side apps and APIs. Its primary goal is to offer a straightforward, adaptable, and neutral framework for creating online applications and managing HTTP requests and responses.

Prerequisite

Node.js

Node.js must be installed Node.js must be installed on your machine because Express.js is built on top of it. The most recent version of Node.js is available for download and installation from the official Node.js website (https://nodejs.org/). Node.js comes with npm (Node Package Manager), which is used to install and manage dependencies for your Express.js projects.

3.3 TOOLS USED

3.3.1 VISUAL STUDIO CODE

Visual Studio Code is a free and open-source code editor developed by Microsoft and available on Windows, Linux, and macOS. It has gained popularity among developers due to its wide range of features including support for various programming languages and syntaxes. The editor is designed to be lightweight and fast, making it suitable for both simple and complex coding tasks. It also includes an integrated terminal, which allows developers to run scripts and commands directly from the editor. Visual Studio Code has a large community of developers who have created various extensions and plugins to enhance its functionality. The versatility and ease of use make it a popular choice among developers across all industries and platforms. Debugging tools, Git integration, and the 's ability to extend its functionality via plugins.

CHAPTER 4

SOLUTION APPROACH

The solution approach for dynamically loading ad components in a website can be broken down into the following steps:

1. Start by creating a new project and setting it up with the necessary dependencies.
2. Design and build the main Web Page that will serve as the user interface. This component will display the content and ads to the user.
3. Create a Content Component that handles content requests from the Web Page. This component will communicate with a Content Component to retrieve the content based on user interactions.
4. Develop Main Component that handles ad requests from the Web Page. And it will send the Ad data only if the collections Start time is matched with the systems Current time.
5. All the Ad collections are stored in the mongoDB in json format .Connect the Main Component to the data base by sending requests for ad data and receiving the responses.
6. Display the retrieved content and ads on the Webpage. Use the data from the Content Component to update the Web Page dynamically. Place the ads received from the Main Component in relevant positions on the page.

INTEGRATION

To integrate the Express.js component for managing image collections into your web page, you can follow the steps below:

1. Begin by including the necessary dependencies for the Express.js component. This may include libraries such as Express.js itself and any additional modules required for image handling or date comparison. You can include these dependencies by linking their CDN files or by downloading and referencing them locally in your HTML file.
2. Define the image collection component by creating the necessary JavaScript code. This can be done either in a separate JavaScript file or within a <script> tag directly in the HTML file. Define the component options, such as the route for fetching collections, handling the start and end dates, and dynamically displaying the appropriate collection based on the current date.
3. Create an instance of the Express.js component. This can be done by including the relevant code in a separate JavaScript file or within a <script> tag placed at the bottom of your HTML file. Instantiate the component, configure any necessary parameters, and attach it to the appropriate element or container within your web page.
4. Insert the Express.js component into the desired location within your HTML template. This is done by adding the component's HTML tag or custom element to the corresponding tag. This allows the component to be rendered and displayed in the specified area of your web page.

TESTING

Retrieval of Active Collections

This test case focuses on the GET request to the "/collections/active" endpoint, aiming to verify the retrieval of active collections. The test sends a GET request using the request object to the specified endpoint. It then checks that the response has a status code of 200, indicating a successful request.

Additionally, the test ensures that the response body contains a property called "collections". Each item in the collections array is expected to have the "isActive" property set to true. This verifies that the endpoint correctly retrieves active collections and returns them in the response body.

Testcase : Passed

Loading of Images from the Active Collections

This test case focuses on retrieving images for active collections. It describes a GET request made to the server for fetching images related to the active collections. The test case verifies that the server responds with a 200 status code, indicating a successful request. It uses the request object to send a GET request to the appropriate endpoint responsible for retrieving images for active collections. After receiving the response, the test asserts that the statusCode property of the response is equal to 200, ensuring that the request was successful and the server is properly handling the retrieval of images for active collections.

Testcase : Passed

Dynamic component

This test case focuses on dynamically loading images from the active collections into a web page using our component. It verifies that the images are being retrieved and displayed correctly.

The test case scenario involves making a GET request to the server, specifically targeting the endpoint responsible for fetching the active collections. The response from the server is then examined to ensure it returns a 200 status code, indicating a successful request.

Within the test, we use the **request** object to send a GET request to the appropriate endpoint. After receiving the response, we check that the **statusCode** property of the response object is equal to 200, confirming that the request was successful.

Testcase : Passed

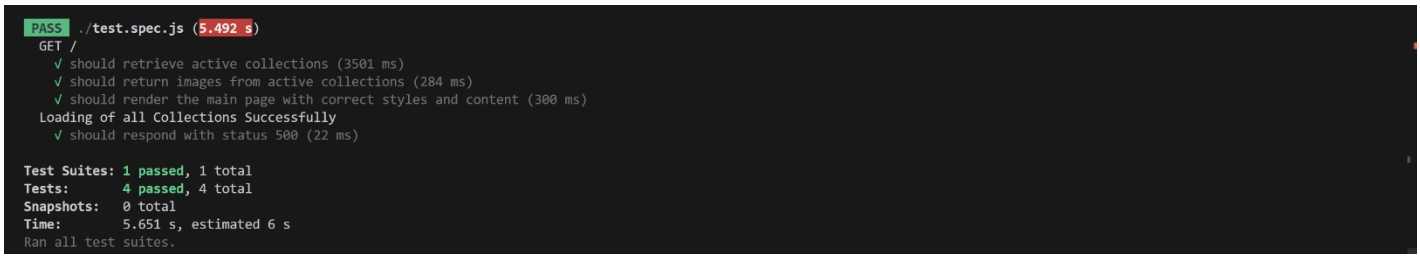

Fig 4.1.1: Overall Testing

SONARQUBE Report

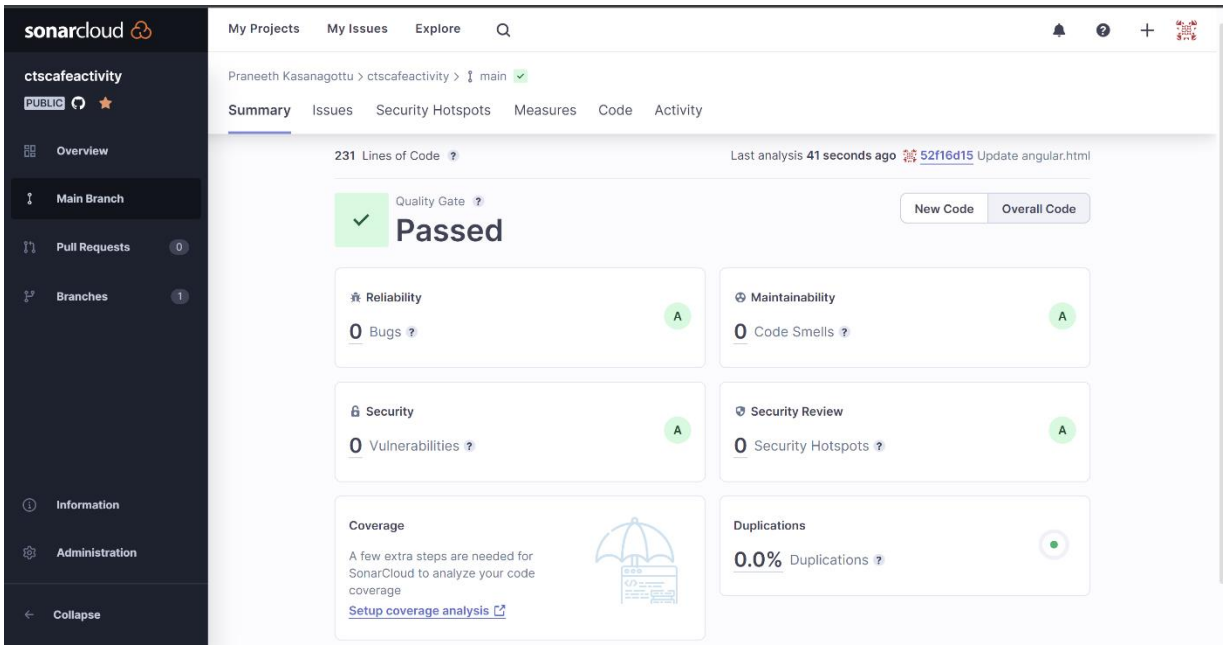

Fig 4.1.2: SonarQube Report

In the project, it achieved a Reliability rating of A, indicating a low number of bugs and no reported security vulnerabilities. Maintainability received an A rating, indicating a clean codebase with no reported code smells. The Security Review revealed no security hotspots or vulnerabilities. Lastly, the Duplications rating is 0.0%, indicating no code duplication. Overall, the project has successfully passed the Quality Gate set by Sonar Cloud.