**Technical Documentation for**

**BRAND front-End Application**

**Table of Contents**

**1. Introduction…………………………………………………………………….. 2**

1.1. Purpose………………………………………………………………………..….. 2

1.2. Audience………………………………………………………………….……… 2

**2. Application Overview………………………………………………………… 2**

**3. Architecture……………………………………………………………………… 2**

3.1. Client-Side Rendering (CSR) ……………………………………………… 2

3.2. API Layer…………………………………………………………………..…….. 2

3.3. Testing Infrastructure………………………………………………..………….. 3

3.4. Deployment ……………………………………………………………………… 3

**4. Prerequisites…………………………………………………………………… 3**

4.1. System Requirements………………………………………………………….. 3

4.2. Clone Repository……………………………………………………………….. 4

4.3. Install Dependencies…………………………………………………………… 4

4.4. Environment Variables………………………………………………………… 4

**5. Configuration………………………………………………………………….. 4**

**6. Development…………………………………………………………………….. 4**

6.1. Folder Structure………………………………………………………………….. 4

6.2. Development Workflow……………………………………………………….. 5

**7. Components………………………………………………………..…………… 6**

**8. API Integration………………………………………………………………….. 7**

8.1. API Endpoints……………………………………………………………………. 7

8.2. API Requests…………………………………………………………………….. 7

**9. Testing…………………………………………………………………………… 7**

**9.1. Test Case 1: Look and Feel Design…………………………………………… 7**

**9.2. Test Case 1: Look and Feel Design…………………………………………… 9**

**9.3. Test Case 1: Look and Feel Design…………………………………………… 9**

**10. License………………………………………………………………………….. 10**

**12. Specification Sign off……………………………………………………………. 11**

**1. Introduction**

**1.1. Purpose**

This technical documentation aims to provide extensive insights and guidance for implementing the Figma design in the front end of the application. It covers development, deployment, maintenance, and usage, offering comprehensive information to facilitate these processes.

**1.2. Audience**

This documentation is intended for developers, contributors, and anyone involved in maintaining or using the application.

**2. Application Overview**

The front-end application system breathes life into the provided Figma designs for the Brand platform, ensuring pixel-perfect accuracy in visualizing user interfaces. It streamlines the conversion of design concepts into digital assets. Key features and functionalities include:

* Implementation of Figma designs for precise visualization of user interfaces.
* A menu of development services, including web and mobile app development, UI/UX design, and more.
* Manual testing for comprehensive quality assurance, encompassing functional testing.

**3. Architecture**

The architecture of our front-end application system ensures flexibility, scalability, and security while efficiently delivering the services mentioned in the application overview.

**3.1. Client-Side Rendering (CSR)**

* **User Interfaces:** The client side is responsible for rendering user interfaces based on Figma designs, including HTML and CSS.
* **Front-End Framework**: The application leverages the React framework for efficient client-side rendering.

**3.2. API Layer**

* **API Endpoints:** The application communicates with the server-side through RESTful API endpoints.

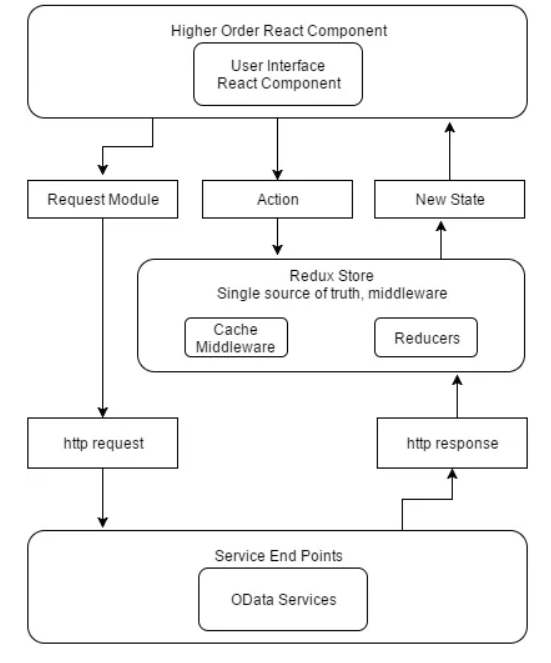
**3.3. Testing Infrastructure**

* **Testing Environment:** The testing environment focuses on manual UI testing, mirroring the production environment.
* **User Interface Evaluation:** Unit testing evaluates user interfaces for functionality, usability, and visual consistency, comparing design specifications to the implementation.

**3.4. Deployment**

* **Deployment to Git:** The front-end application is deployed to Git, emphasizing version control, collaboration, and code management.

**3.5. Architectural Diagram**

****

**4. Prerequisites**

**4.1. System Requirements**

* Operating System: Windows, macOS, or Linux.
* Web Browsers: Chrome, Firefox, Safari, or Edge for testing.
* Version Control: Git.
* Package Managers: React.
* Web Server: Tomcat.
* Mobile Emulators or Devices: Android Emulator and iOS Simulator.
* Graphic Design Software: Figma and Adobe Photoshop.
* Stable Internet Connection.
* Sufficient RAM and Storage Space.

**4.2. Clone Repository**

To clone a Git repository, install Git, navigate to the desired directory, and use the `git clone` command.

**4.3. Install Dependencies**

Install project dependencies using npm (Node Package Manager) by navigating to the project directory and running `npm install`.

**4.4. Environment Variables**

Define environment variables in a `.env` file for configuration. Use `REACT\_APP\_` prefix in React applications.

**5. Configuration**

The `.env` or `.env.local` file is used for managing environment variables in your React application.

**6. Development**

**6.1. Folder Structure**

- `node\_modules`: Contains project dependencies.

- `public`: Stores static assets.

- `public/index.html`: Main HTML file.

- `public/documentations`: Store Technical specifications and README

- `public/assets`: For images and media.

- `public/assets/css`: CSS or SCSS files.

- `public/assets/style.css`: Style management.

- `public/assets/img`: Images and media.

- `src/components`: JavaScript files.

- `src/App.js`: Main application component

- `src/index.js`: Entry point for React app.

**6.2. Development Workflow**

**Step 1: Clone the project**

* Clone the project repository from the Git repository to your local development environment. Follow the provided instructions for cloning the repository.



**Step 2: Navigate to the project directory:**



**Step 3: Environment Configuration**

* Ensure you have the necessary software and tools installed as indicated in 4.1.
* Install project dependencies using npm.



* Set up environment variables (if applicable) by creating a .env file in the project root and defining variables in the KEY=VALUE format.



**Step 4: Folder Structure**

* The project follows a structured folder hierarchy indicated in 6.1.

**Step 5. Coding Standards**

* Adhere to established coding standards and guidelines, including consistent indentation, proper naming conventions (camelCase or PascalCase), and descriptive comments.
* Utilize JavaScript ES6/ES7 features consistently.
* Organize files and directories logically for maintainability.

**Step 6: Coding**

* Write clean, modular, and maintainable code.
* Use React components for UI elements and organize them logically.
* Follow the established folder structure and coding standards for consistency.

**Step 7: Testing**

* Perform manual testing of the application's user interfaces to ensure functionality, usability, and visual consistency.
* Evaluate the application's responsiveness on various devices and screen sizes.
* Verify that the menu functionality works as expected.
* Test the application's error handling by introducing errors in API responses.

**Step 8: Deployment**

* Deploy updates to staging environments for thorough testing before production deployment.
* Ensure that only stable and tested code is deployed to the production environment.

**Step 9: Documentation**

* Maintain comprehensive documentation for the project, including README files with setup instructions, usage guidelines, and information about project dependencies.
* Document APIs, components, and other significant aspects of the application.
* Document Technical specification

**Step 10: User Interface (UI)**

* Follow responsive design principles to ensure the application functions well on various devices and screen sizes.
* Maintain a consistent and visually pleasing UI design.
* Implement common UI/UX design patterns for navigation, forms, modals, and notifications.
* Enhance user experience with subtle micro-interactions, animations, and typography.

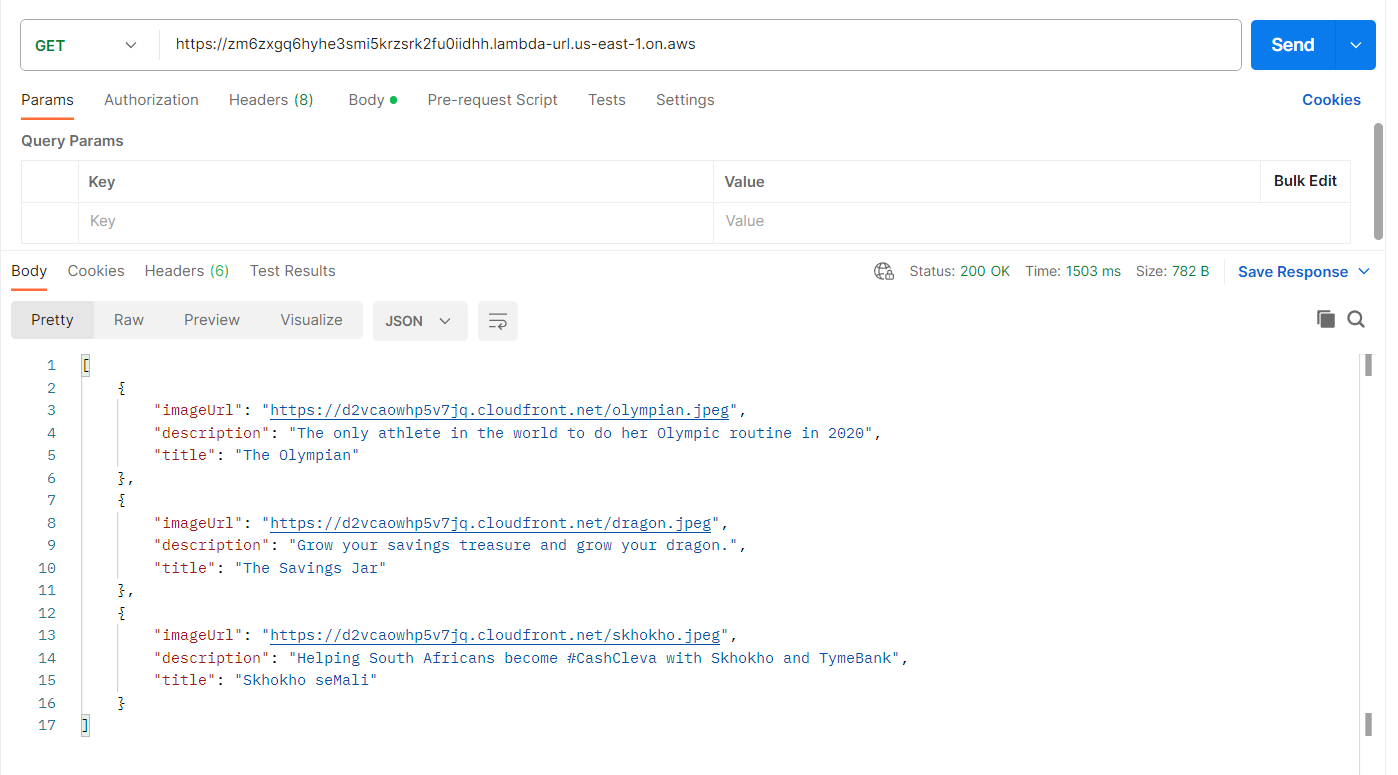
**7. Components**

* **Header Component:** This component is responsible for rendering the application's header, which typically includes the site logo, navigation links, and possibly user-related actions.
* **Footer Component:** The footer component handles the display of the application's footer section. It often contains links to the terms of service, privacy policy, and contact information.
* **Navigation Component:** The navigation component takes care of rendering navigation links and menus within the application. It can handle user authentication and display different links based on the user's status.
* **Hero Section Component:** This component is responsible for rendering the hero section of the application, which usually includes a large, attention-grabbing image or video along with a call-to-action or important messages.
* **Carousel Component:** the carousel component is used to create dynamic image carousels or sliders that allow users to view multiple images or pieces of content in a rotating manner.
* **Image Gallery Component:** The image gallery component is responsible for displaying collections of images, often in a grid or with options for users to view larger versions of the images.

**8. API Integration**

**8.1. API Endpoints:** <https://zm6zxgq6hyhe3smi5krzsrk2fu0iidhh.lambda-url.us-east-1.on.aws>

**8.2. API Requests**



**9. Testing**

Manual UI testing is performed. Document test cases and results.

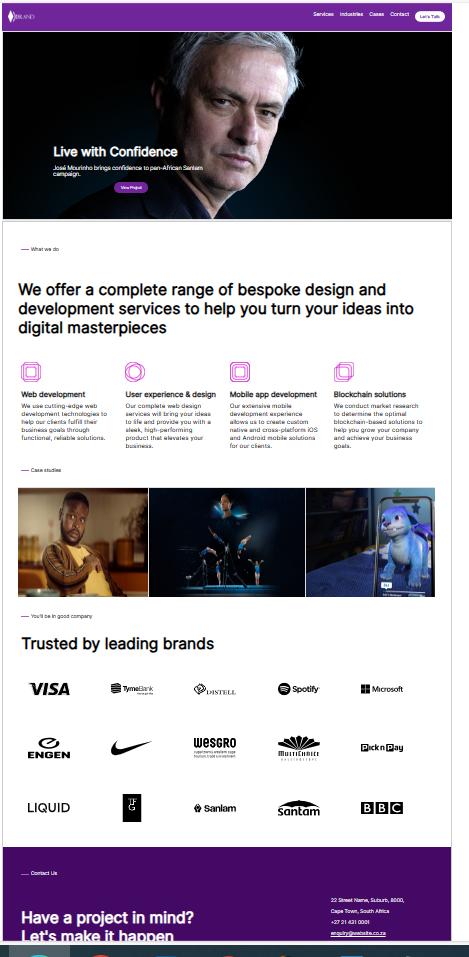
**9.1. Test Case 1: Look and Feel Design**

Test Description: Ensure that the application's UI matches the provided Figma design for the Brand front-end.

Test Steps:

* Open the application in a supported web browser.
* Navigate to the home page.
* Compare the UI elements, layout, and styling with the Figma design.
* Expected Result: The application's UI should closely match the Figma design, including colors, fonts, layout, and visual elements.

Actual Result: The application's UI matches the Figma design, and no visual discrepancies are observed.



Pass/Fail: Pass

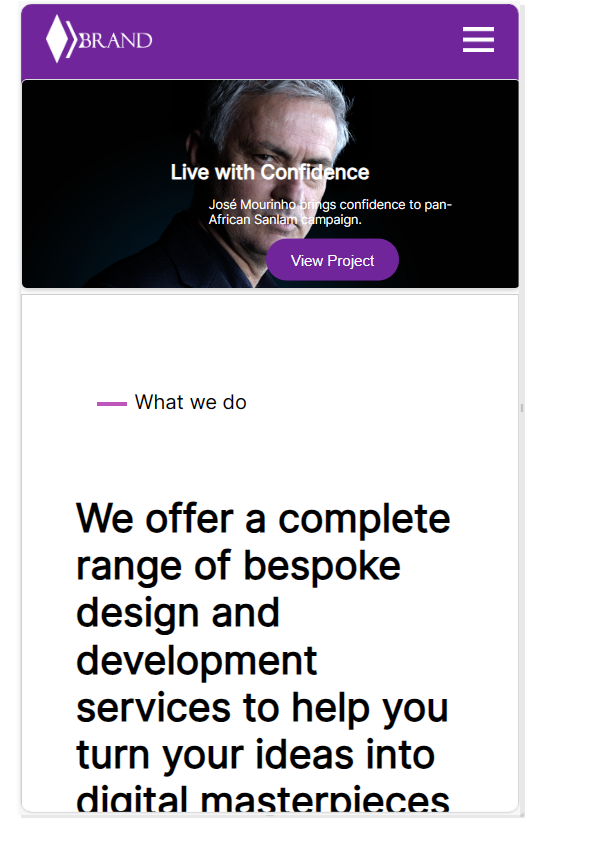
**9.2. Test Case 2: Responsiveness**

Test Description: Verify that the application is responsive and functions correctly on various devices and screen sizes.

Test Steps:

* Open the application in different web browsers (e.g., Chrome, Firefox, Safari, Edge).
* Resize the browser window to simulate different screen sizes (desktop, tablet, mobile).
* Test various features and interactions on each device.
* Expected Result: The application should adapt to different screen sizes, and all features should work as expected on different devices.

Actual Result: The application is responsive and functions correctly on various devices and screen sizes.



Pass/Fail: Pass

**9.3. Test Case 7: Cross-Browser Compatibility**

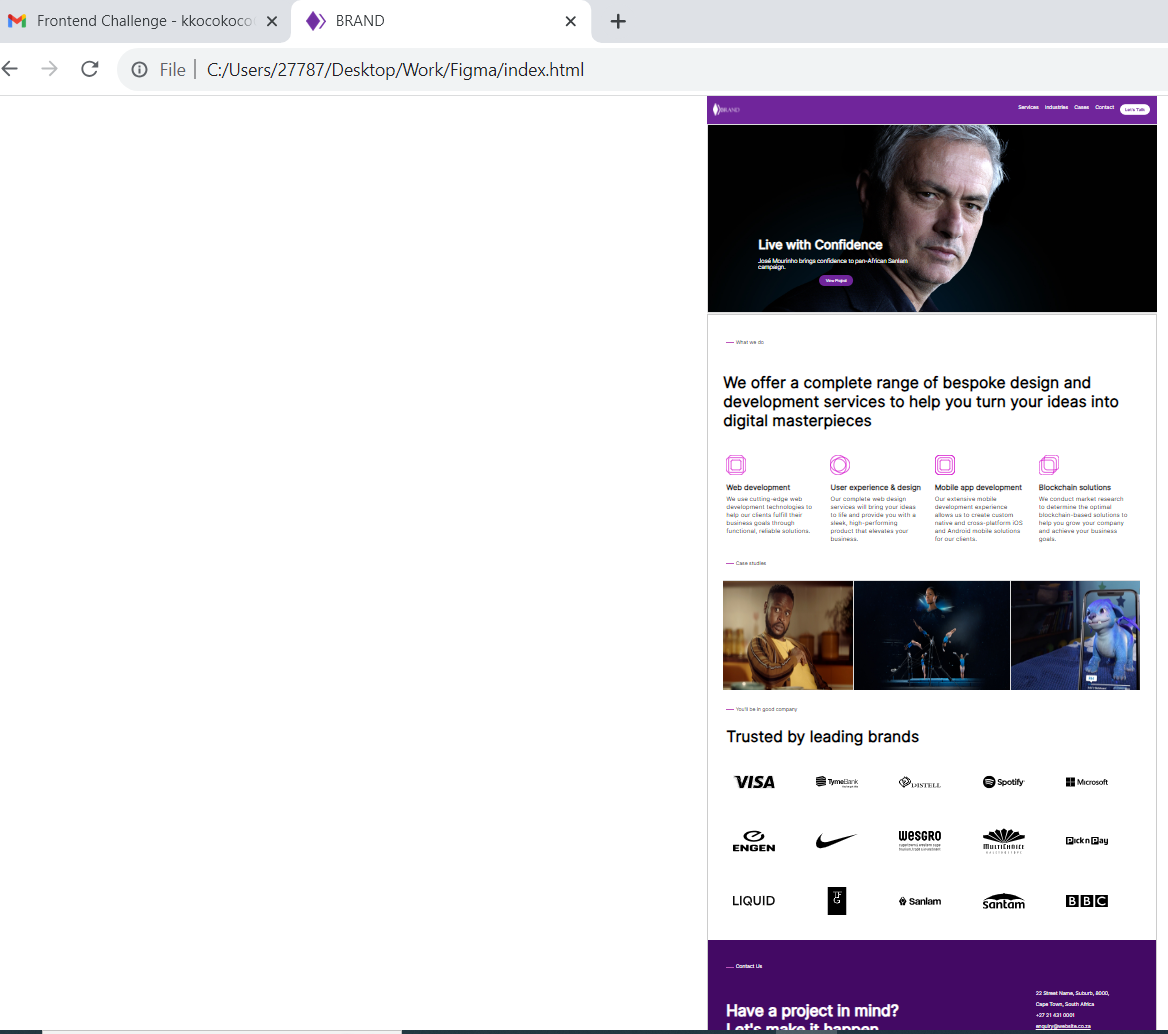
Test Description: Ensure that the application functions correctly on different web browsers.

Test Steps:

* Test the application on various web browsers, including Chrome, Firefox, Safari, and Edge.
* Verify that all features work as expected on each browser.
* Expected Result: The application should be compatible with major web browsers, and all features should work without issues.

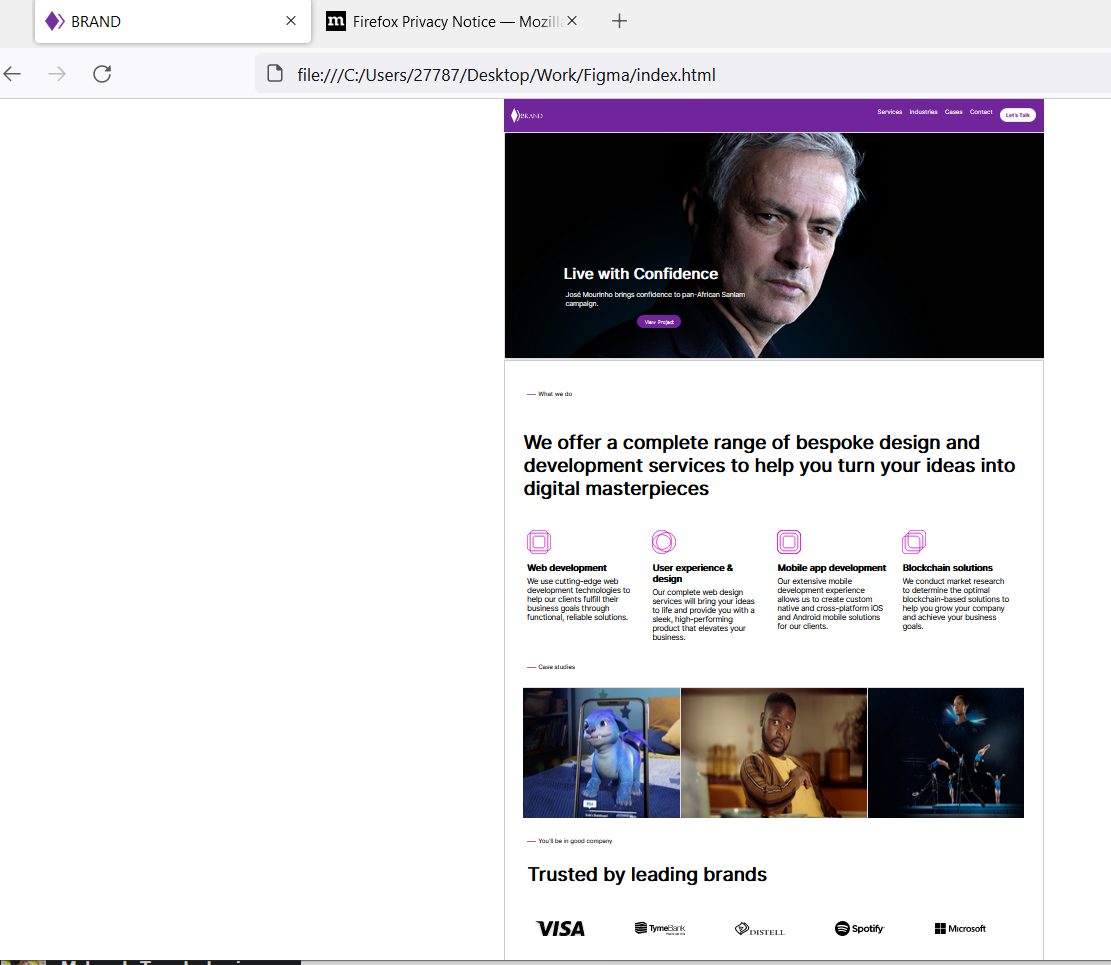
Actual Result: The application functions correctly on all tested web browsers.

For Chrome browser



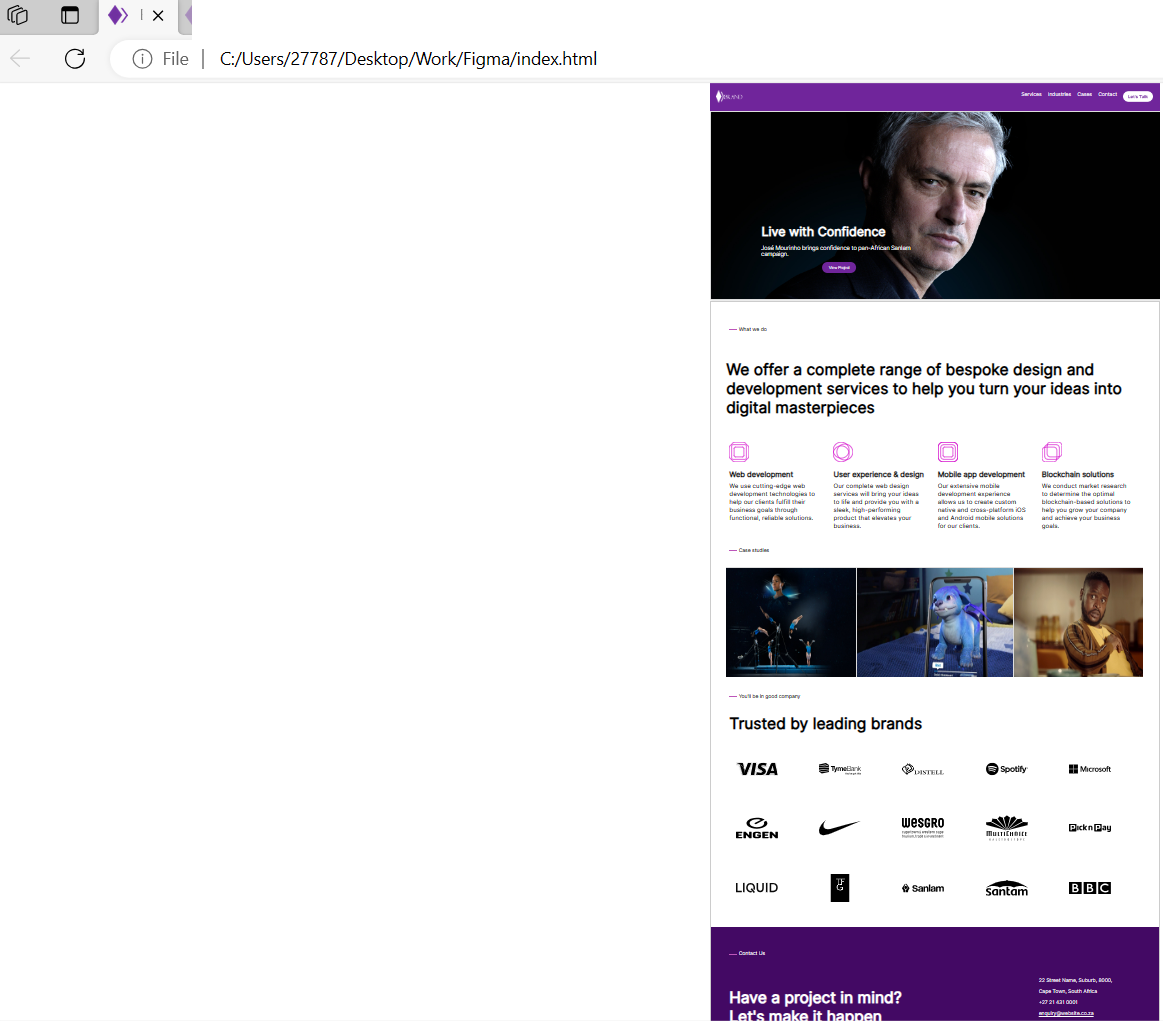
Pass/Fail: Pass

Results for Firefox



Pass/Fail: Pass

Results for Edge



Pass/Fail: Pass

**10. License**

No specific license is required, as open-source technologies are used.

**11. Specification Sign-off**

Provide contact details and signatures for project maintainers/developers.

Developer: Tsundzuka­­ni Mabunda

Date 23-09-2023

Signature ------------------------