**Software Design Document**

### **1. Title and People**

* **Title**: Comprehensive Website Optimization and Analysis Platform
* **Status**: In progress
* **Team Members**:
  + @Ankita
  + @Vidya
  + @Animesh

### **2. Context**

* **Background**: The market for SEO and website optimization tools is saturated with high-priced platforms that are often too complex for small businesses to navigate. This creates a barrier for small businesses to effectively optimize their websites and compete in the digital space.
* **Relevance**: Our platform aims to bridge this gap by providing an affordable, user-friendly solution that empowers small businesses to optimize their websites without the need for extensive technical knowledge.

### **3. Goal**

Our goal is to deploy the platform with all planned features, ensuring that website quality aligns with these features. By doing so, users will experience firsthand that the platform’s features are not only effective but also actively followed by the platform itself.

### **4. Problems**

* **Limited Expertise**: Our team members possess a foundational understanding of this field, but we acknowledge that there’s room for improvement. We’re committed to expanding our knowledge and addressing the problem diligently.
* **Dependency on External Services**: Some features rely on external platforms and APIs. If cost or integration becomes an obstacle, we’ll carefully evaluate their inclusion in our platform. Balancing functionality with practicality is crucial.
* **Innovative Features**: Certain features are groundbreaking and lack established theories or off-the-shelf solutions. For these, we’ll need to design custom algorithms from scratch. It’s an exciting challenge, but it requires creativity and persistence.

### **5. Market Need**

* **Market Analysis**: The SEO and website optimization market is growing, with increasing demand from small businesses looking to improve their online presence. However, the high cost and complexity of existing solutions create a significant barrier.
* **Target Audience**: Small businesses and startups with limited budgets and technical expertise.
* **Pain Points**: High costs, complexity of existing platforms, and the need for user-friendly, affordable solutions.

### **6. Competitor Analysis**

For each feature of our platform, we have conducted a detailed analysis of existing competitors, identifying their strengths, weaknesses, and opportunities for improvement. Below is a brief overview with example:

For **Color Grading** competitors like Colorfyit and ColorFinder Playground analyze color usage but lack impact grading. We see an opportunity to develop a tool that not only analyzes color usage but also grades the impact of color combinations on UI/UX.

For complete details, please refer to the full [competitor analysis document.](https://docs.google.com/document/d/1JQ43DYk2tqJV2SzkqE3gPqpevYzfSnRIZFyn1xPFuiY/edit?usp=sharing)

### **7. Product Details**

Our platform offers a comprehensive suite of features designed to optimize and analyze websites effectively. Below is a list of key features. Click on each feature to get complete details, including its importance, how it works, how to use it, work distribution, references, technical elements, and more.

#### **Table of Features with Progress**

| **Features** | **Frontend** | **Backend** | **Frontend Handler** | **Backend**  **Handler** |
| --- | --- | --- | --- | --- |
| [Color Grading](https://docs.google.com/document/d/1Mkd1U1K7lBYEoU995Ed4cEFA7LqP5T_5qJLeiwScYi4/edit) | 10% | 60% | Vidya | Animesh |
| [Content Style Grading](https://docs.google.com/document/d/17j9y3xZhKEqeJ61nm_P2E2rwe3C9SmRU_vbUhnG0jj4/edit#heading=h.rwxxbvm9jdug) | 10% | 60% | Ankita | Animesh |
| [Elements Layout Measurement](https://docs.google.com/document/d/1Uloi1mdoaLqAInvwsvlswYNcIco5ZJk_DATecOL1V2Y/edit#heading=h.5uzh792930ko) | 10% | 50% | Animesh | Vidya |
| [Web Page Performance](https://docs.google.com/document/d/1Io2me3HcvRHL5oxTL0sXNqaLUukXVYNnSOqa8i7IqsY/edit) | 0% | 60% | Vidya | Animesh |
| [Web Page Quality](https://docs.google.com/document/d/1N44K921OwCFZ5QV1zNPetKKV1QfG8xwFfVIJaWlTReQ/edit#heading=h.49egndh6zllo) | 0% | 30% | Ankita | Animesh |
| [Leveling User Interaction](https://docs.google.com/document/d/1GG84WUO_Bdxy57ypXQZrqZ8EdaCMRrLbmL2ymbvFSxM/edit#heading=h.rsmbf8cvv1ar) | 0% | 0% | Animesh | Animesh |
| [Website Security and Policy](https://docs.google.com/document/d/1qWXR3bIeSNIwqNap7q4zzL9NjPbpE5AhpvZ9t7t6JzQ/edit#heading=h.mzujllu8h9ez) | 0% | 80% | Vidya | Vidya |
| [Threats Mitigation](https://docs.google.com/document/d/1NDog9oO4gK19zfO6McL81CxNYPy-PvkeJHRPfQA36p0/edit#heading=h.25y7cimeoy3y) | 0% | 15% | Vidya | Vidya |
| [Calculating Website Ranking](https://docs.google.com/document/d/1GQbgSTwVU0aHmp_1veGst516Nu6i7D2wPx-KQOPhrks/edit) | 0% | 50% | Ankita | Ankita |
| [SEO Grading](https://docs.google.com/document/d/1YfzjAcJs3SvDNma-vUqp-LozhBMagsNDIZxXoirSlfI/edit#heading=h.9qhza15e8r4) | 0% | 90% | Ankita | Ankita |
| [Competitor Analysis](https://docs.google.com/document/d/1SdOWMltT4ZxZ6S24uuY-CxqIFvA3Gn72fcFZNtwNk8M/edit#heading=h.9c51hx4ky4o4) | 0% | 0% | Vidya | Vidya |
| [Keyword Tracking](https://docs.google.com/document/d/1aAgP3s3OVbLEc31jt09jreKLxoXoPFdAnLk70SglM0s/edit) | 0% | 0% | Ankita | Ankita |
| [Trend Analysis](https://docs.google.com/document/d/1HMIfz6lwFIaiMrWPtQLQhH6v3V2JOpVC0hRVE5uIpd4/edit) | 0% | 0% | Ankita | Ankita |
| [Ad Recommendation](https://docs.google.com/document/d/1uWClqlI8YsuM4U4aB9VWvmL9ajbrO9n4D_OOQrrg3co/edit) | 0% | 90% | Vidya | Vidya |
| Ad Positioning | 0% | 0% |  |  |
| Website Marketing | 0% | 0% |  |  |

### 

### 

### **8. User Interface (UI) Design**

#### **Mockups**

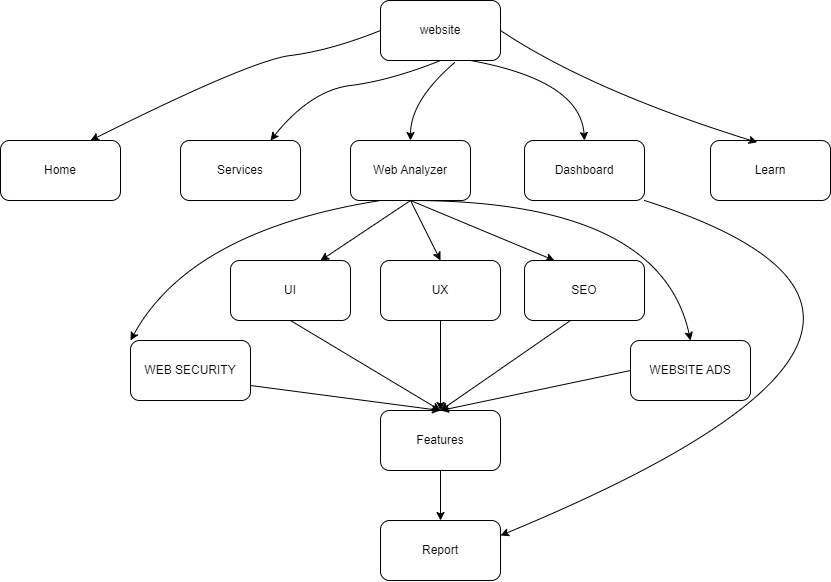
We have created wireframes and prototypes for the key pages of our platform to ensure a user-friendly and intuitive interface. You can view the wireframes and prototypes for each page by clicking on the links below:

* **Home Page**: [Wireframe](https://drive.google.com/file/d/1kQT6itqatSJBZzha263uq4pfRv-R2INw/view?usp=sharing) and [Prototype](https://drive.google.com/file/d/1-lQu3hHZPEx6PBuE6YxYOhEdC8KtYGip/view?usp=sharing) ([Link](https://play.teleporthq.io/projects/spotless-hungry-crocodile-y52b6d/preview/TQ_Pv0bI6un8i))
* **WebAnalyzer Page**: [Wireframe](https://drive.google.com/file/d/136JsoOIFVlukNbIstV_o6spTpy--2_9P/view?usp=sharing) and Prototype
* **Learn Page**: [Wireframe](https://drive.google.com/file/d/10OquhHeUItlDVFeev03RYZrgfDRLzd7Z/view?usp=sharing) and [Prototype](https://drive.google.com/file/d/1LmPe1S1RelTsg0Ksj77wZRwwHn6Md_XU/view?usp=sharing) ([Link](https://play.teleporthq.io/projects/spotless-hungry-crocodile-y52b6d/preview/TQ_XBU0dlEQsG))
* **Dashboard Page**: [Wireframe](https://drive.google.com/file/d/1gpkEYEYz6DRLI9g-Tc5ezb2UIy8wt9Rg/view?usp=sharing) and [Prototype](https://drive.google.com/file/d/1fNwdX3mfMYtFor0RHZrvisXkbH2SQhgV/view?usp=sharing) ([Link](https://play.teleporthq.io/projects/user-dashboard-fptyzr/preview/TQ_Yvwn_NgZ_F))
* **Report Page:** Wireframe and Prototype

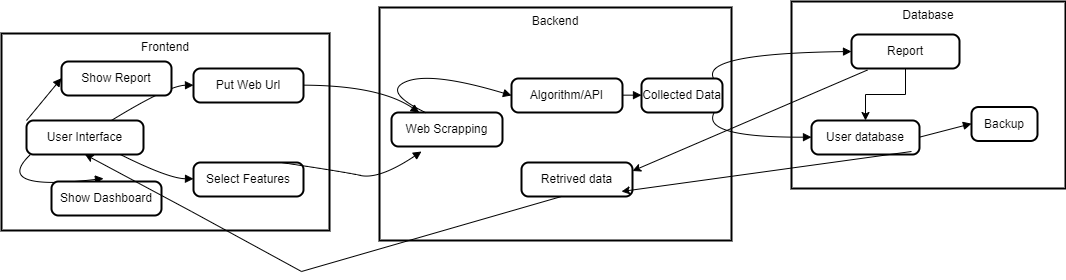
#### **User Flows**

User flows describe how users will interact with the system, from the initial entry point to completing their tasks. Below are the diagrams illustrating the website flow and higher-level component flow:

* **Website Flow Diagram**: This diagram outlines the logical path a user takes when navigating through the website, from the homepage to various features like WebAnalyzer, Learn, and Dashboard.



* **Higher-Level Component Flow Diagram**: This diagram provides a macro-level perspective of the system architecture, highlighting the major components, their interactions, and the flow of data between them.



### **9. Technology Stack**

#### **Frontend Technologies**

* **React**: Used for building the user interface. React allows for the creation of reusable UI components and ensures a responsive and dynamic user experience.
* **HTML, CSS, JavaScript**: Core web technologies used for structuring, styling, and adding interactivity to the web pages.
* **Bootstrap**: A CSS framework used for designing responsive and mobile-first web pages.

#### **Backend Technologies**

* **FastAPI**: A modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.
* **Flask**: A lightweight WSGI web application framework in Python. Used for building web applications and APIs.

#### **Data Visualization**

* **Chart.js**: A JavaScript library used for creating interactive and responsive charts and graphs.

#### **Web APIs**

* **Web APIs**: Used for integrating various external services and data sources into the platform.

#### **Web Scraping Tools**

* **Web Scraping Tools**: Used for extracting data from websites to provide insights and analysis.

#### **Machine Learning Models**

* **Machine Learning Models**: Used for analyzing data and providing predictive insights. These models will be integrated into the backend to enhance the platform’s analytical capabilities.

#### **Prompt Engineering**

* **Prompt Engineering**: Techniques used to design and optimize prompts for AI models to improve their performance and relevance.

#### **Backend Algorithms**

* **Backend Algorithms**: Custom algorithms developed to handle specific tasks such as data processing, analysis, and optimization.

### **10. API Design**

#### **Endpoints**

1. **GET /api/v1/home**
   * **Purpose**: Retrieve the home page content.
   * **Parameters**: None
2. **GET /api/v1/web-analyzer**
   * **Purpose**: Retrieve web analyzer results.
   * **Parameters**: url (string) - The URL of the website to analyze.
3. **GET /api/v1/learn**
   * **Purpose**: Retrieve learning resources.
   * **Parameters**: None
4. **GET /api/v1/dashboard**
   * **Purpose**: Retrieve dashboard data.
   * **Parameters**: userId (string) - The ID of the user.
5. **GET /api/v1/report**
   * **Purpose**: Retrieve report data.
   * **Parameters**: reportId (string) - The ID of the report.
6. **POST /api/v1/web-analyzer/submit**
   * **Purpose**: Submit data for web analysis.
   * **Parameters**:
     + url (string) - The URL of the website to analyze.
     + feature (string) - The specific feature to analyze (e.g., color grading, content style).
7. **POST /api/v1/newsletter/subscribe**
   * **Purpose**: Subscribe to the newsletter.
   * **Parameters**:
     + email (string) - The email address to subscribe.
     + name (string, optional) - The name of the subscriber.

#### **Data Formats**

* **Requests**: JSON
* **Responses**: JSON

### **11. Security Considerations**

#### **Threat Mitigation**

* **Potential Threats**: SQL injection, Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Data breaches.
* **Mitigation Strategies**:
  + Use parameterized queries to prevent SQL injection.
  + Implement input validation and sanitization to prevent XSS.
  + Use anti-CSRF tokens to prevent CSRF attacks.
  + Encrypt sensitive data both in transit and at rest.

#### **Data Protection**

* **Encryption**: Use TLS for data in transit and AES-256 for data at rest.
* **Access Control**: Implement role-based access control (RBAC) to restrict access to sensitive data.
* **Regular Audits**: Conduct regular security audits and vulnerability assessments.

### **12. Performance Considerations**

#### **Performance Metrics**

* **Response Time**: Measure the time taken to respond to API requests.
* **Throughput**: Measure the number of requests handled per second.
* **Error Rate**: Measure the percentage of failed requests.
* **Latency**: Measure the delay before a transfer of data begins following an instruction.

#### **Optimization**

* **Caching**: Implement caching strategies to reduce load times.
* **Load Balancing**: Use load balancers to distribute traffic evenly across servers.
* **Database Optimization**: Optimize database queries and indexing.
* **Code Optimization**: Regularly review and optimize code for performance improvements.

### **13. Testing Plan**

#### **Test Cases**

* **Unit Tests**: Test individual components and functions.
* **Integration Tests**: Test the interaction between different components.
* **End-to-End Tests**: Test the entire application flow from start to finish.
* **Performance Tests**: Test the system’s performance under various conditions.
* **Security Tests**: Test for vulnerabilities and security flaws.

#### **Testing Tools**

## **Unit Testing**

### **PyTest (Python)**

* **Usage**: Ideal for testing individual components of the backend services built with FastAPI and Flask.
* **How It Works**: Write test functions to validate the behavior of specific functions or methods. PyTest can run these tests and report any failures, ensuring that each unit of the code works as expected.

### **Jest (JavaScript)**

* **Usage**: Perfect for testing React components and JavaScript functions.
* **How It Works**: Jest write tests for the React components, ensuring they render correctly and handle state changes properly. It also supports mocking and snapshot testing.

## **Integration Testing**

### **Postman**

* **Usage**: Useful for testing API endpoints of the backend services.
* **How It Works**: Create and run collections of API requests to ensure that different parts of the application work together as expected. Postman can automate these tests and provide detailed reports.

### **Selenium**

* **Usage**: Suitable for testing the integration of the frontend and backend.
* **How It Works**: Selenium automates browser actions to simulate user interactions with the web application. We can write scripts to test the flow of data between the frontend and backend, ensuring that the entire system works seamlessly.

## **End-to-End Testing**

### **Cypress**

* **Usage**: Excellent for testing the entire user journey on the platform.
* **How It Works**: Cypress allows us to write tests that simulate real user interactions, from visiting a page to performing actions like clicking buttons and filling out forms. It ensures that the application works correctly from start to finish.

## **Performance Testing**

### **JMeter**

* **Usage**: Ideal for load testing of the backend services.
* **How It Works**: JMeter can simulate multiple users accessing the application simultaneously, measuring how well the backend handles the load. It helps identify performance bottlenecks and optimize server response times.

### **Locust**

* **Usage**: Useful for performance testing with a focus on user behavior.
* **How It Works**: Locust defines user behavior in Python code and simulates thousands of users interacting with the application. It provides insights into how the system performs under different load conditions.

## **Security Testing**

### **OWASP ZAP (Zed Attack Proxy)**

* **Usage**: Essential for identifying security vulnerabilities in the web application.
* **How It Works**: OWASP ZAP can scan the application for common security issues like SQL injection, XSS, and more. It provides detailed reports and recommendations for fixing vulnerabilities.

### **Burp Suite**

* **Usage**: Comprehensive tool for security testing.
* **How It Works**: Burp Suite performs manual and automated security testing. It includes features like a proxy server, scanner, and intruder to identify and exploit vulnerabilities in your application.

### **14. Deployment Plan**

#### **Environment**

* **Hardware Requirements**: Servers with sufficient CPU, memory, and storage.
* **Software Requirements**: Operating systems, web servers, databases, and other necessary software.

#### **Deployment Steps**

1. **Setup Environment**: Configure servers and install necessary software.
2. **Deploy Backend**: Deploy FastAPI and Flask applications.
3. **Deploy Frontend**: Deploy React application.
4. **Configure Database**: Set up and configure the database.
5. **Integrate APIs**: Connect external APIs and services.
6. **Run Tests**: Execute all test cases to ensure functionality.
7. **Monitor Deployment**: Monitor the system for any issues post-deployment.

### **15. Maintenance Plan**

#### **Monitoring**

* **Tools**: Use monitoring tools like Prometheus, Grafana, and ELK Stack to monitor system performance and health.
* **Metrics**: Monitor key metrics such as CPU usage, memory usage, response times, and error rates.

#### **Updates**

* **Process**: Follow a structured process for updates, including development, testing, staging, and production deployment.
* **Frequency**: Regularly update the system to include new features, improvements, and security patches.

### **16. Reasoning and Justification**

#### **Design Decisions**

* **React for Frontend**: Chosen for its component-based architecture and efficient rendering.
* **FastAPI and Flask for Backend**: Selected for their performance, ease of use, and flexibility.
* **Chart.js for Data Visualization**: Provides interactive and responsive charts.
* **Machine Learning Models**: Enhance analytical capabilities and provide predictive insights.

#### **Market Fit**

* **Affordable Pricing**: Addresses the budget constraints of small businesses.
* **User-Friendly Interface**: Ensures both technical and non-technical users can effectively use the platform.
* **Essential Features**: Focuses on critical aspects without unnecessary complexity.

#### **Competitive Advantage**

* **Cost-Effective**: More affordable than many existing SEO platforms.
* **Simplicity**: Easier to navigate and use compared to competitors.
* **Comprehensive Analysis**: Provides detailed insights and actionable recommendations.

### **17. Appendices**

#### **Glossary**

* **API**: Application Programming Interface
* **SEO**: Search Engine Optimization
* **UI**: User Interface
* **RBAC**: Role-Based Access Control
* **NLP**: Natural Language Processing
* **OWASP**: Open Web Application Security Project

### **18. Future Plans:**

### Next Semester Deep Learning Project: Advanced features like ad positioning, web page quality, keyword, competitor, and trend analysis will be part of the next semester's project, as they require machine learning models.

### Learning Page: A dedicated page will be added to help users understand SEO parameters, utilizing prompt engineering for better guidance.

### Next.js Migration: For enhanced flexibility and performance, the website's SEO system will shift to Next.js in the future.

### 