Website Accessibility Checklist

UI Design Document

Team 2: Jacob Mathison, Emily Buzle, Taylor Williams, Jadyn Smith, Javid Ditty

Table of Contents

1.0 Introduction

1.1 Goals and Objectives

The goal of this software is to facilitate the assessment process of a website by using an accessibility checklist in the form of a questionnaire. It will allow the users to evaluate the quality and effectiveness of an app. This will be achieved through a website that allows users to answer questions regarding the organization and presentation of content, the functionality of website components, efficiency of use, ability to find relevant information, as well as overall aesthetics. These questions will be evaluated and interpreted to provide the user with a score that shows the quality of the accessibility of the website.

1.2 Statement of Scope

Users

Users of this software will be web application testers and end users (referenced as users in this document) that will evaluate website accessibilities with a questionnaire.

Inputs

Users will provide “yes,” “no,” and “n/a” answers to each question in the accessibility checklist.

Processing Functionality

User responses will be stored temporarily in order to compute the necessary calculations to determine the score. But results and responses will not be saved long-term. This means that the only data storage will be for the questions. This will be stored in a JSON file.

Outputs

There will be three pages for this application. The first will be the user input screen where the questions will be presented, described, and answered. Also on this screen will be the title and a short description of the software scope and functionality.

The second screen will be the results screen which will describe the methods and equations used to find the totals for each section of the questionnaire. The final total will be presented as well as final insights based on the score. There will also be a page for the user manual as well, which will answer any questions the user may have and display the calculations used to find the resulting score.

1.3 Software Context

This questionnaire will allow for the easy quantification of the quality of the accessibility of a web application. For users and developers testing and providing feedback on applications that they are using or making, this provides a consistent source of feedback that can be replicated. This makes the testing and reviewing process more objective and allows people to return to our software and take the questionnaire again to gauge improvement on their development. We will use the Web Content Accessibility Guidelines (WCAG), specifically where it applied to web applications. This is a set of documents formulated by the Accessibility Guidelines Working Group which covers recommendations for making web content more accessible, and many of the points covered apply to apps by extension. Another source that we will use as a guide to formulate our questions in order to evaluate the software of other mobile applications is the WC3 Mobile Application Best practices. It is different from the mobile browsing best practices because it considers the development of web applications for mobile devices, which is what our accessibility checklist will be evaluating.

1.4 Major Constraints

One constraint is related to the number and types of questions that the user is required to answer to determine the quality of their application’s accessibility. The development team will determine the types of questions that need to be answered to determine the state of the user’s application.

The questionnaire will be also limited to yes, no and n/a questions. Each question will need to provide hints that will elaborate the question for the user to help them determine if their application answers the question or not. Each hint will only provide guidance on the current question. For each question, the user only has to click the yes or no option to answer the question.

A weighted table will be designed to calculate how important each question is to create an accessible application. It will be constrained based on what the developers determine is important to create an accessible application. The rating that the user will be given on a scale of bad, fair, good, great or excellent with the overall score.

2.0 Data Design

2.1 Internal Software Data Structure

Question – Model that represents a question that can be put into the checklist.	
Attribute/Method	**Definition**
- QUESTIONS: Object	Questions read from the database file using read(path). For information about its structure, see Section 2.4.
- id: int	Unique number that is used to identify a question.
- content: String	That which is being asked by a question.
- answer: boolean	That which is given in response to a question.
- weight: int	Impact that the question has on the accessibility score.
- hint: String	Elaboration on the content of a question.
- tip: String	Guide for following guidelines that a question implies.
+ constructor(id: int)	Constructs a question object and initializes it with the question that has the id in QUESTIONS.
- read(path: String): void	Reads object at database file located at path and stores the result in QUESTIONS. For information about its structure, see Section 2.4.

2.2 Global Data Structure

ChecklistHandler – Controller that handles the checklist and its interactions.	
Attribute/Method	**Definition**
- questions: Question[]	Questions that are on the checklist.
+ constructor()	Constructs a ChecklistHandler object, initializing its question field using its getQuestions() method.
+ getQuestions(): Question[]	Converts the QUESTIONS attribute in the Question class into Question objects.

+ sendQuestions(): void	Sends questions to the ChecklistView class to render.
+ getScore():int	Calculates the accessibility score from each question.answer in ChecklistHandler.questions and returns it.
+ sendScore(score: int): void	Sends score to the ResultView class to render.
+ sendTips(ids: String[]): void	Retrieves tips from the questions with an id in ids and sends them to the ResultView class to render.
+ getNo(): Question[]	Gets each question in ChecklistHandler.questions with a “No” answer and returns them.
+ getAnswers(): void	Updates each question.answer in ChecklistHandler.questions with the corresponding answer in questionnaire.
+ submit(href: String): void	Gets the accessibility score and the ids of all the questions with “No” answers; attaches them as query strings to href; and directs the user to the resulting url.
+ clear(): void	Send a request to ChecklistView to clear the questionnaire.

ChecklistView – View that represents the checklist.

Attribute/Method	Definition
+ renderQuestions(questions: { id:String : {content: String, hint: String } }): void	Renders the UI with all the questions in questions.
+ clear(): void	Resets all the answer radio button groups in the questionnaire to the “N/A” answer.

ResultView – View that represents the checklist results.

Attribute/Method	Definition
+ renderScore(score: int, rating: int): void	Renders the accessibility score, the accessibility rating, and the score bar progress with score and rating.

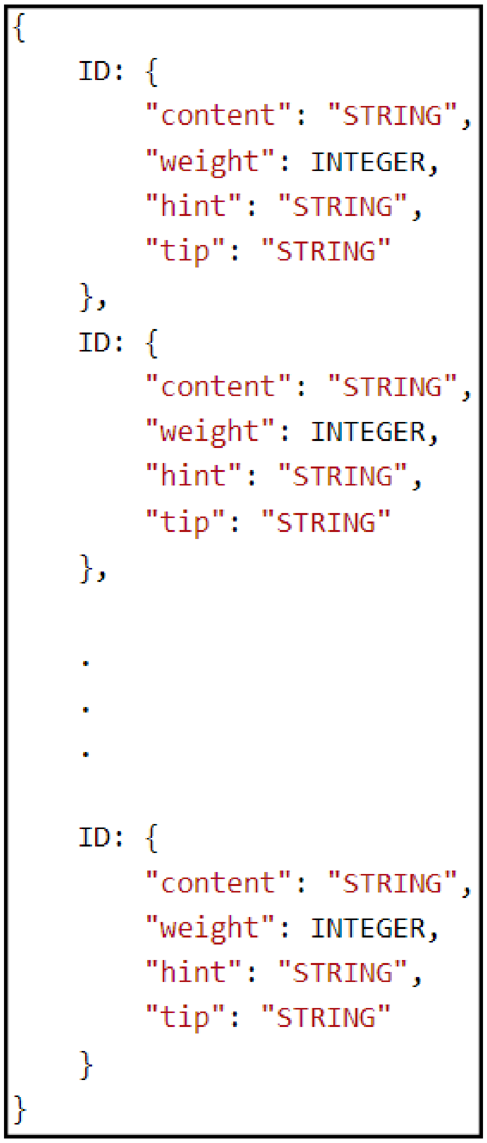
+ renderTips(tips: String[]): void	Renders the tips section with all the tips in tips.

2.3 Temporary Data Structure

No temporary data structures are used in this project.

2.4 Database Description

Because only question ids, contents, weights, hints, and tips require permanent storage and JSON files have a high compatibility with JavaScript, this project will use a JSON file as a database. The file will have the format shown in the image titled "2.4 - Database Format" with the id as the primary key.



```
{
    ID: {
        "content": "STRING",
        "weight": INTEGER,
        "hint": "STRING",
        "tip": "STRING"
    },
    ID: {
        "content": "STRING",
        "weight": INTEGER,
        "hint": "STRING",
        "tip": "STRING"
    },

    .
    .
    .

    ID: {
        "content": "STRING",
        "weight": INTEGER,
        "hint": "STRING",
        "tip": "STRING"
    }
}
```

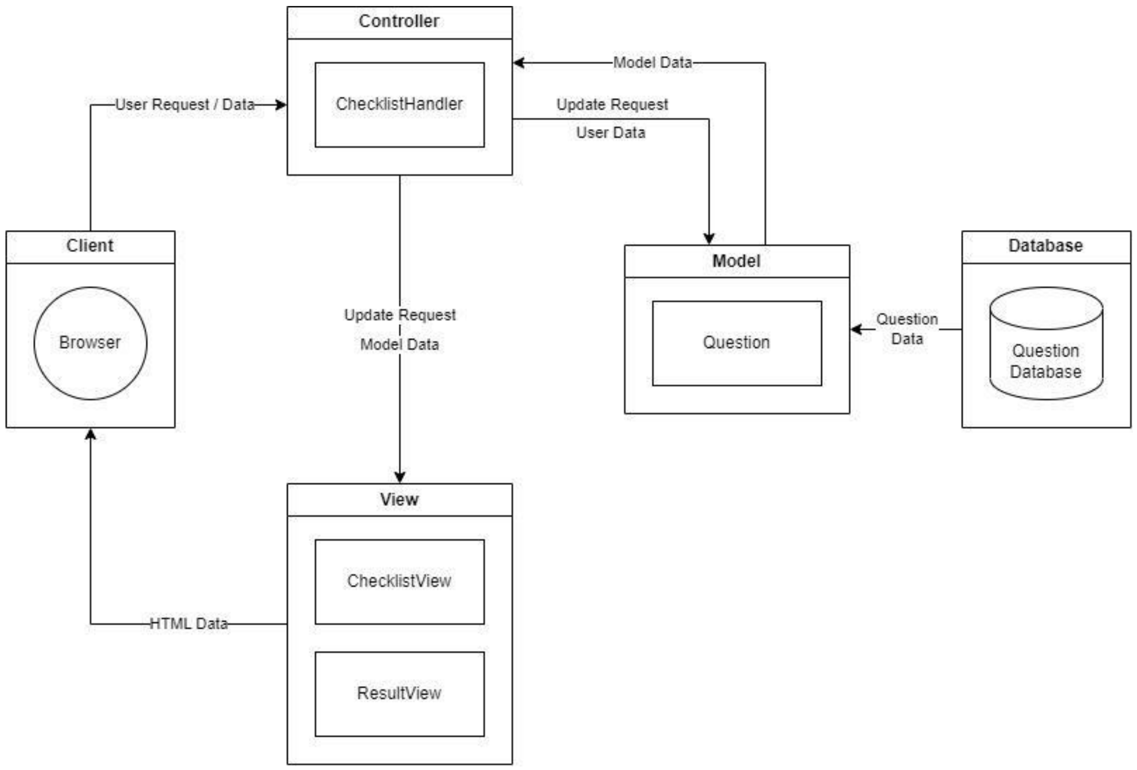
2.4 - Database Format

3.0 Architectural and Component-Level Design

3.1 Program Structure

For this project, we have selected the MVC (Model View Controller) Architecture. The Model component contains content and logic concerning the manipulation of data, and it has sole access to the database. The View component handles the visual aspect of the website, or the UI, and it portrays the data model to the user. The Controller component connects the Model and View, handling the real-time data requests from the browser. For us, this means that it will handle answer collection, and the calculation of the accessibility score, values that will never be stored within the database.

3.1.1 Architecture Diagram



3.1.2 Alternatives

We selected the MVC architecture because:

- It is a popular web development architecture.
- It separates the project into three distinct components that will improve unit testing and maintenance tasks.
- It reduces the coupling between components, improving the reusability of each component.

We considered alternative architectures such as:

- *Data-Centered* – Because our project relies on a database that contains questions, a data store is central to our project. However, data-centered architectures are designed for projects that frequently access and update data within the store. Our project does not modify the store directly, and it only accesses it once at startup, contradicting the purpose of this architecture.
- *Object-Oriented* – Because our project's functions can be captured by distinct and well-defined objects and their communication, this architecture could have been a good choice for our project. However, it does not define object roles as distinctly as the MVC architecture, making it more difficult to understand and maintain.

3.2 Description of Components

3.2.1 Question

3.2.1.1 Processing Narrative (PSPEC)

Before instantiation, the Question class initializes its QUESTIONS field with the object contained in the database file. QUESTIONS is initialized using the read method. Instantiation starts once a question id is passed to the Question class constructor. The constructor retrieves the data from QUESTIONS that have a matching id, creates a question object, and initializes the id, content, weight, hint, and tip fields of the question object with the data. The answer field is initialized with a null value.

3.2.1.2 Interface Description

Question receives as input an id from a ChecklistHandler object and an id, content, weight, hint, and tip from the database file. Question sends as output its id, content, weight, hint, answer, and tip fields to the ChecklistHandler.

3.2.1.3 Description of Sub-Components

3.2.1.3.1 constructor(id: int)

3.2.1.3.1.1 Interface Description

It receives as input an id from a ChecklistHandler object and content, weight, hint, and tip from Question.QUESTIONS. It sends as output a Question object.

3.2.1.3.1.2 Algorithmic Model

```
Function constructor(id: int)
        this.id = id
        this.content = Question.QUESTION[id].content
        this.weight = Question.QUESTION[id].weight
        this.hint = Question.QUESTIONS[id].hint
        this.tip = Question.QUESTIONS[id].tip
        this.answer = null
End
```

3.2.1.3.1.3 Restrictions/Limitations

To operate correctly, the database file must exist, and the system must be able to access it. It is used to define the QUESTIONS field.

3.2.1.3.1.4 Local Data Structures

It uses the QUESTIONS, id, content, weight, hint, tip, and answer fields.

3.2.1.3.2 read(path: String): Object

3.2.1.3.2.1 Interface Description

It receives as input a path to the database file with the structure described in section 2.4. It sends as output the object it extracts from the database file.

3.2.1.3.2.2 Algorithmic Model

```
Function read(path: String): void
        json = null
        read object from file at path into result
        Question.QUESTIONS = json
End
```

3.2.1.3.2.3 Restrictions/Limitations

To operate correctly, the database file must exist, and the system must be able to access it.

3.2.1.3.2.4 Local Data Structures

It uses QUESTIONS and the object structure defined in section 2.4.

3.2.1.3.2.5 Performance Issues

Because this sub-component reads data from a file on the hard drive, the speed of this reading depends on the structure and health of the drive itself as well as the drivers used to read it.

3.2.2 ChecklistHandler

3.2.2.1 Processing Narrative (PSPEC)

Instantiation starts without any arguments passed to the ChecklistHandler class constructor. The constructor retrieves the QUESTIONS attribute in the Question class using the getQuestions method and converts it into a collection of Question objects, storing them in the questions field. Upon receiving requests from the browser, ChecklistHandler objects can send questions, accessibility scores, and tips to the views to be rendered using sendQuestions, sendScore, and sendTips respectively. They can also update the answers of their questions field with the answers received from the browser.

3.2.2.2 Interface Description

ChecklistHandler receives as an input the object in the QUESTIONS attribute in the Question class and question answers from the browser. It sends as an output questions, accessibility scores, and tips to the views to be rendered using sendQuestions, sendScore, and sendTips respectively.

3.2.2.3 Description of Sub-Components

3.2.2.3.1 constructor()

3.2.1.3.1.1 Interface Description

It receives nothing as an input but updates its questions field as an output with questions from its getQuestions() method.

3.2.2.3.1.2 Algorithmic Model

```
Function constructor()
        this.questions = this.getQuestions();
End
```

3.2.2.3.1.3 Local Data Structures

It uses the QUESTIONS attribute in the Question class.

3.2.2.3.2 getQuestions(): Question[]

3.2.1.3.2.1 Interface Description

It receives as an input the object in the QUESTIONS attribute in the Question class. It sends as an output Question objects to its questions field.

3.2.2.3.2.2 Algorithmic Model

```
Function getQuestions(): Question[]
```

```
    for each key in Question.QUESTIONS
        questions[key] = Question(key)
End
```

3.2.2.3.2.3 Local Data Structures

It uses the QUESTIONS attribute in the Question class.

3.2.2.3.3 sendQuestions(): void

3.2.2.3.3.1 Interface Description

It receives nothing as an input, and it sends as an output question ids, contents, and hints to the ChecklistView object.

3.2.2.3.3.2 Algorithmic Model

```
Function sendQuestions(): void
    questions = {}
    [For each question in ChecklistHandler.question, append id :
    {content, hint} to questions]
    ChecklistView.renderQuestions(questions)
End
```

3.2.2.3.3.3 Local Data Structures

It uses the questions attribute in the ChecklistHandler class.

3.2.2.3.4 getScore(): int

3.2.2.3.4.1 Interface Description

It receives nothing as an input, and it sends as an output an accessibility score.

3.2.2.3.4.2 Algorithmic Model

```
Function getScore(): int
    let yes, total = 0;
    For each question in ChecklistHandler.questions
        if (question.answer != null) total += question.weight
        if (question.answer === true) yes += question.weight
    return (total === 0) ? 100 : (100 * yes) / total
End
```

3.2.2.3.4.3 Local Data Structures

It uses the questions attribute in the ChecklistHandler
class.

3.2.2.3.5 sendScore(score: int, rating: int): void

3.2.2.3.5.1 Interface Description

It receives a score and rating as input, and it sends as an output an accessibility score to the ResultView object.

3.2.2.3.5.2 Algorithmic Model

```
Function sendScore(score: int, rating: int): void
        rating = “”
        Calculate the rating from score and store it in rating
        ResultView.renderScore(score, rating)
End
```

3.2.2.3.6 sendTips(ids: String[]): void

3.2.2.3.6.1 Interface Description

It receives question ids as input, and it sends as an output question tips to the ResultView object.

3.2.2.3.6.2 Algorithmic Model

```
Function sendTips(ids: String[]): void
        tips = [];
        for each question in ChecklistHandler.questions with id in ids
                tips.push(question.tip);
        ResultView.renderTips(tips)
End
```

3.2.2.3.6.3 Local Data Structures

It uses the questions attribute in the ChecklistHandler
class.

3.2.2.3.7 getNo(): Question[]

3.2.2.3.7.1 Interface Description

It receives nothing as input, and it sends as an output questions in ChecklistHandler with “No” answers.

3.2.2.3.7.2 Algorithmic Model

```
Function getNo(): Question[]
    no = []
    for each question in questions with answer === false
        no.push(question);
    return no;
End
```

3.2.2.3.7.3 Local Data Structures

It uses the questions attribute in the ChecklistHandler
class.

3.2.2.3.8 getAnswers(): void

3.2.2.3.8.1 Interface Description

It receives nothing as input, and it sends as an output updated answers to questions in ChecklistHandler.questions.

3.2.2.3.8.2 Algorithmic Model

```
Function getAnswers(): void
    for each answer in the document
        assign answer to corresponding question in questions
End
```

3.2.2.3.8.3 Local Data Structures

It uses the questions attribute in the ChecklistHandler
Class.

3.2.2.3.9 submit(href: String): void

3.2.2.3.9.1 Interface Description

It receives a href as an input, and it sends as an output a navigation request to the browser.

3.2.2.3.9.2 Algorithmic Model

```
Function submit(href: String): void
    getAnswers()
    score = getScore()
    no = getNo()
    url = `${href}?score=${score}&tips=`
    Append each question id in no to url's query string
    Navigate browser to url
End
```

3.2.2.3.9.3 Local Data Structures

It uses the questions attribute in the ChecklistHandler class.

3.2.2.3.10 clear(): void

3.2.2.3.10.1 Interface Description

It receives nothing as input and returns a render request to ChecklistView to reset answers in the document to “N/A.”

3.2.2.3.10.2 Algorithmic Model

```
Function clear(): void
    ChecklistView.clear();
End
```

3.2.3 ChecklistView

3.2.3.1 Processing Narrative (PSPEC)

ChecklistView is not supposed to be instantiated; all its operations are on the class, rather than the object, level. It receives as inputs an object filled with question ids, contents, and hints. It outputs question UI elements to the browser.

3.2.3.2 Interface Description

It receives as inputs an object filled with question ids, contents, and hints from the ChecklistHandler. It outputs question UI elements to the browser.

3.2.3.3 Description of Sub-Components

3.2.3.3.1 renderQuestions(questions: {id:String {content: String, hint: String}}): void

3.2.3.3.1.1 Interface Description

It receives as inputs an object filled with question ids, contents, and hints from the ChecklistHandler. It outputs question UI elements to the browser.

3.2.3.3.1.2 Algorithmic Model

```
Function renderQuestions(questions: {id:String {content: String,
hint: String}}): void
        html = “”
        for each id in questions
                html += html for questions[id] (See section 4)
        Add html to the questions section of the UI
End
```

3.2.3.3.1.3 Local Data Structures

It uses an object filled with question ids, contents, and hints from the ChecklistHandler

3.2.3.3.2 clear(): void

3.2.3.3.2.1 Interface Description

It receives nothing as input, and it resets answer radio buttons on the UI to “N/A” as output.

3.2.3.3.2.2 Algorithmic Model

```
Function clear(): void
        Select all the answer radio buttons on the document
        Set the “N/A” radio buttons and unset all other buttons
End
```

3.2.4 ResultView

3.2.4.1 Processing Narrative (PSPEC)

ResultView is not supposed to be instantiated; all its operations are on the class level, rather than the object. It receives as inputs an accessibility score and tips. It sends as output accessibility score and tip UI elements to the browser.

3.2.4.2 Interface Description

It receives as inputs an accessibility score and tips. It sends as output accessibility score and tip UI elements to the browser.

3.2.4.3 Description of Sub-Components

3.2.4.3.1 renderScore(score: int): void

3.2.4.3.1.1 Interface Description

It receives as inputs an accessibility score. It sends as output accessibility score UI elements to the browser.

3.2.4.3.1.2 Algorithmic Model

```
Function renderScore(score: int): void
    Set the text accessibility score on the document to score.
    Set the accessibility score bar on the document to score.
End
```

3.2.4.3.2 renderTips(tips: String[]): void

3.2.4.3.2.1 Interface Description

It receives as input question tips. It sends as output tip UI elements to the browser.

3.2.4.3.2.2 Algorithmic Model

```
Function renderTips(tips: String[]): void
    html = “”
    For each tip in tips
        html += html for tip
    Add html to the tips section in the document.
End
```

3.3 Software Interface Description

3.3.1 External Machine Interfaces

This software system will only have a few optional external machine interfaces. In order for a user to configure the software system with ease then serial interfaces can be connected. If the user decides to access the system through a computer then such serial interfaces can be keyboard and mouse devices. However, these serial interfaces are not required to manipulate the system as there are alternatives, such as an on-screen keyboard that generally are pre-installed on computer devices and touchpads that are also usually built onto laptop devices.

3.3.2 External System Interfaces

This software system will mainly function on its own website. The website in question would have to be accessed through an operating system, then to an internet browser. Examples of available operating systems that can be used to access this system can possibly be Microsoft Windows or Mac OS. Upon the user obtaining an operating system, they can navigate to the software's website by using an internet browser such as Google Chrome, FireFox, or Microsoft Edge. Each of these example internet browsers have search engines like Google or Yahoo, where the user can search for the system's website URL if they choose to do so. For this to be functional, the system must be able to connect to a server and/or the internet. The user will be responsible for connecting to a local internet server.

3.3.3 Human Interface

This software will primarily feature a human interface. This specific interface being a user interface that allows the user to easily interact with the software system's features. The user interface will consist of three components: homepage, results page, and manual page. The homepage will display the accessibility checklist the user can fill out. The results page that will show the user how well their system fits in terms of accessibility. Lastly, the manual page will provide information to the user on how to work the accessibility checklist if they need help. More information on human interfaces are available in section 4.0.

4.0 User Interface Design

The user interface in this project will consist of 3 screens. The home screen will be where the user inputs their answers into the accessibility questionnaire. The results screen will output the user's accessibility score calculated from their answers and give some tips to improve their application's accessibility. The third screen is a help manual screen where information will be provided to the user on how to use the questionnaire and the website.

4.1 Description of the User Interface

Home Page

The home page starts with the application title at the top. The button for accessing the help manual page is on the top right of the home page. Below the title is a text that has instructions for using the hints option for each question. Below that are the questions that the user will have to answer to generate an accessibility score.

The question section will contain yes/no questions for the user to answer based on their website. The user can hover over the hint icon to get help with answering that specific question. The user can also select the N/A option to skip answering that specific question.

Below the questions section there are two buttons at the bottom of the page. The clear button will clear all answers in the questionnaire and reset them to N/A. The submit button will signal the application to calculate the user's answer and then send the user to the output page.

Results Page

The output page also has the website application and the help manual button at the top of the page. Below that is the primary output container for the web application. The first output is the calculated accessibility score based on the user's answers on the previous page. Below that is a bar that will give the user an idea of how good they scored on accessibility. Below that is the tips section where tips will be presented to the users based on their no answers in the questionnaire. There is also a home button below the container to take the user back to the home page.

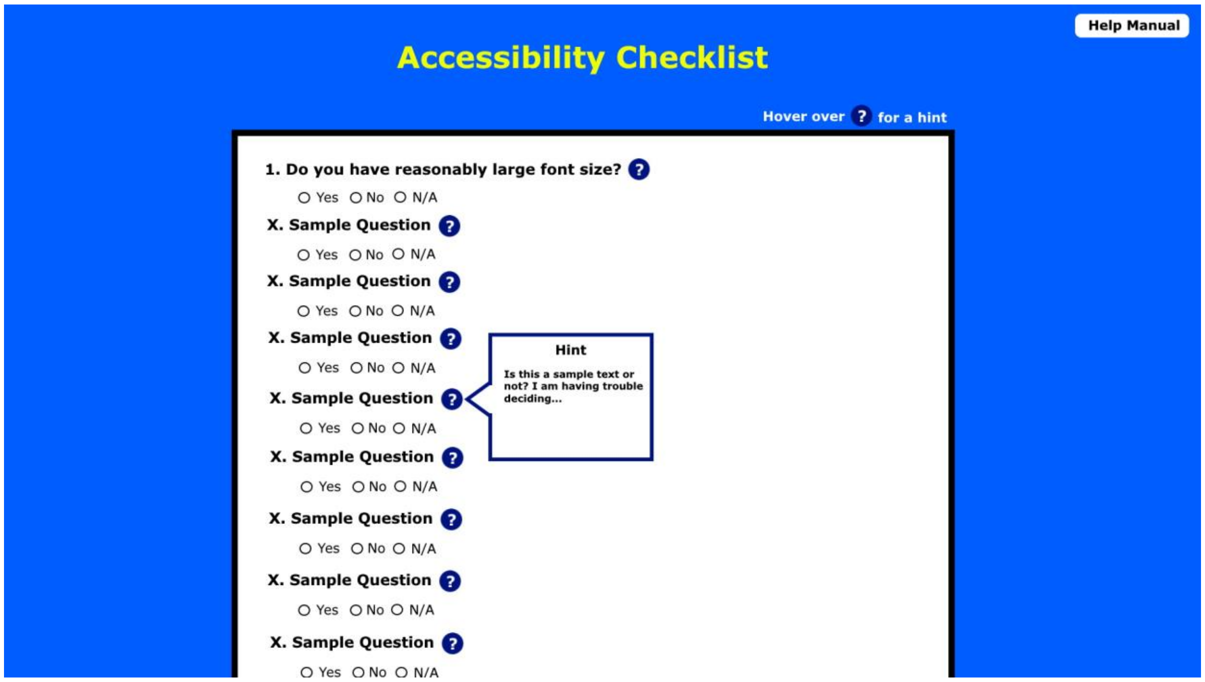
Manual Page

The manual page will contain the application title again, and it will also have a container for application help. The container will have different sections that will instruct the user in how to use the application, particularly the questionnaire, and the website. Below the help container is a home button to return to the home page.

4.1.1 Screen Images

UI Concept Images

Home Page Concept with Hints



Help Manual

Accessibility Checklist

Hover over ? for a hint

1. Do you have reasonably large font size? ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

X. Sample Question ?

O Yes O No O N/A

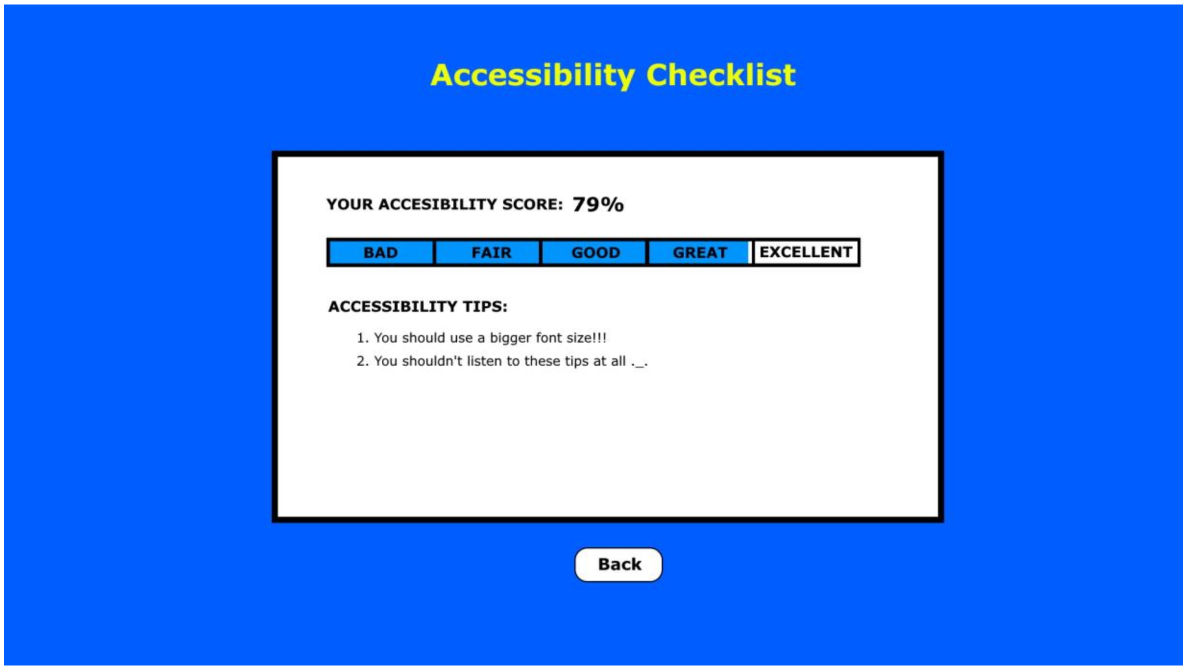
X. Sample Question ?

O Yes O No O N/A

Hint

Is this a sample text or not? I am having trouble deciding...

Results Page Concept



Accessibility Checklist

YOUR ACCESIBILITY SCORE: 79%

BAD	FAIR	GOOD	GREAT	EXCELLENT

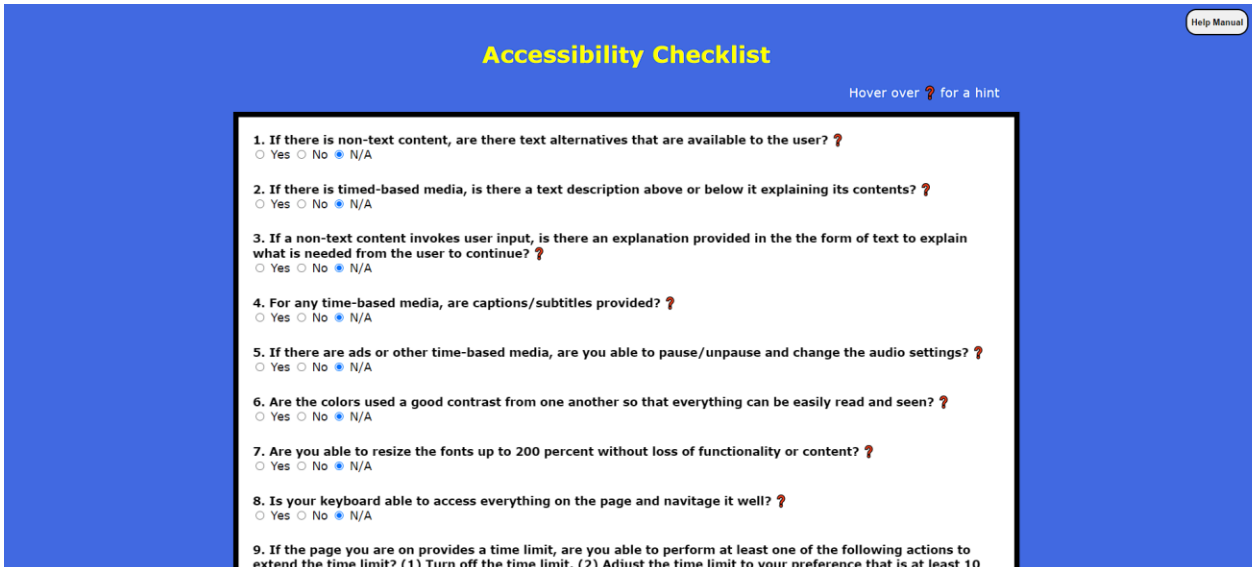
ACCESSIBILITY TIPS:

1. You should use a bigger font size!!!
2. You shouldn't listen to these tips at all ._.

Back

Actual UI Images

Top of Home Page



Accessibility Checklist

Hover over ? for a hint

1. If there is non-text content, are there text alternatives that are available to the user? ?
Yes No [✓ N/A]

2. If there is timed-based media, is there a text description above or below it explaining its contents? ?
Yes No [✓ N/A]

3. If a non-text content invokes user input, is there an explanation provided in the the form of text to explain what is needed from the user to continue? ?
Yes No [✓ N/A]

4. For any time-based media, are captions/subtitles provided? ?
Yes No [✓ N/A]

5. If there are ads or other time-based media, are you able to pause/unpause and change the audio settings? ?
Yes No [✓ N/A]

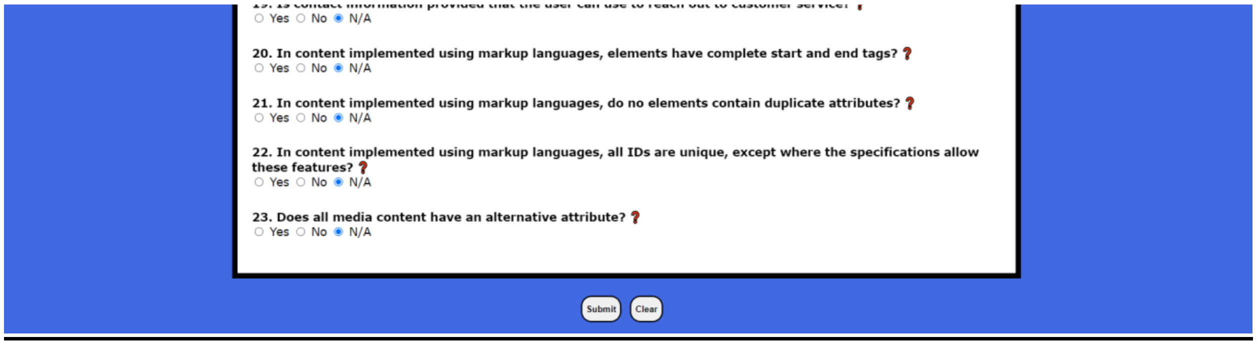
6. Are the colors used a good contrast from one another so that everything can be easily read and seen? ?
Yes No [✓ N/A]

7. Are you able to resize the fonts up to 200 percent without loss of functionality or content? ?
Yes No [✓ N/A]

8. Is your keyboard able to access everything on the page and navitage it well? ?
Yes No [✓ N/A]

9. If the page you are on provides a time limit, are you able to perform at least one of the following actions to extend the time limit? (1) Turn off the time limit. (2) Adjust the time limit to your preference that is at least 10

Bottom of Home Page



19. Is contact information provided that the user can use to reach out to customer service? ?
Yes No [✓ N/A]

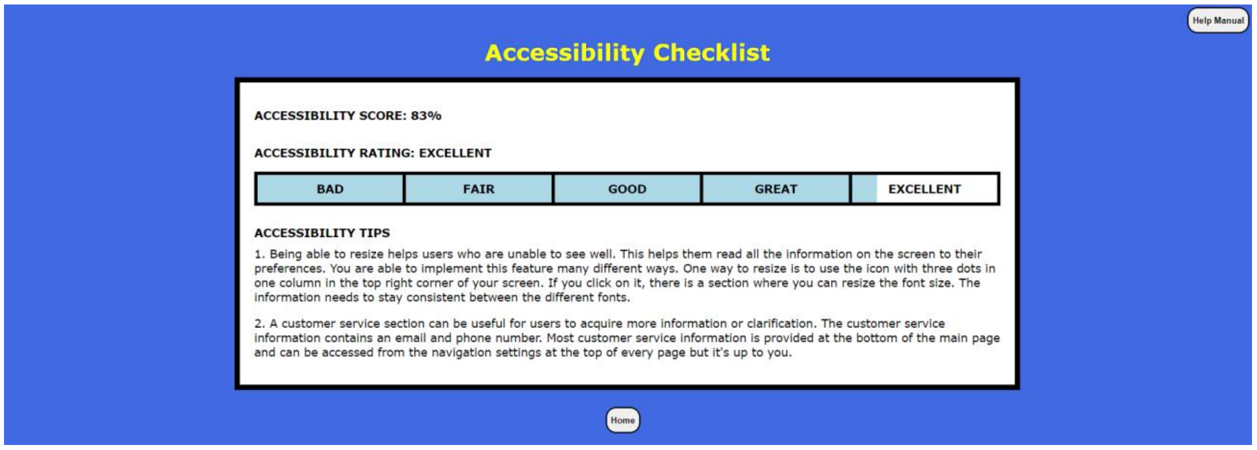
20. In content implemented using markup languages, elements have complete start and end tags? ?
Yes No [✓ N/A]

21. In content implemented using markup languages, do no elements contain duplicate attributes? ?
Yes No [✓ N/A]

22. In content implemented using markup languages, all IDs are unique, except where the specifications allow these features? ?
Yes No [✓ N/A]

23. Does all media content have an alternative attribute? ?
Yes No [✓ N/A]

Results Page



Accessibility Checklist

ACCESSIBILITY SCORE: 83%

ACCESSIBILITY RATING: EXCELLENT

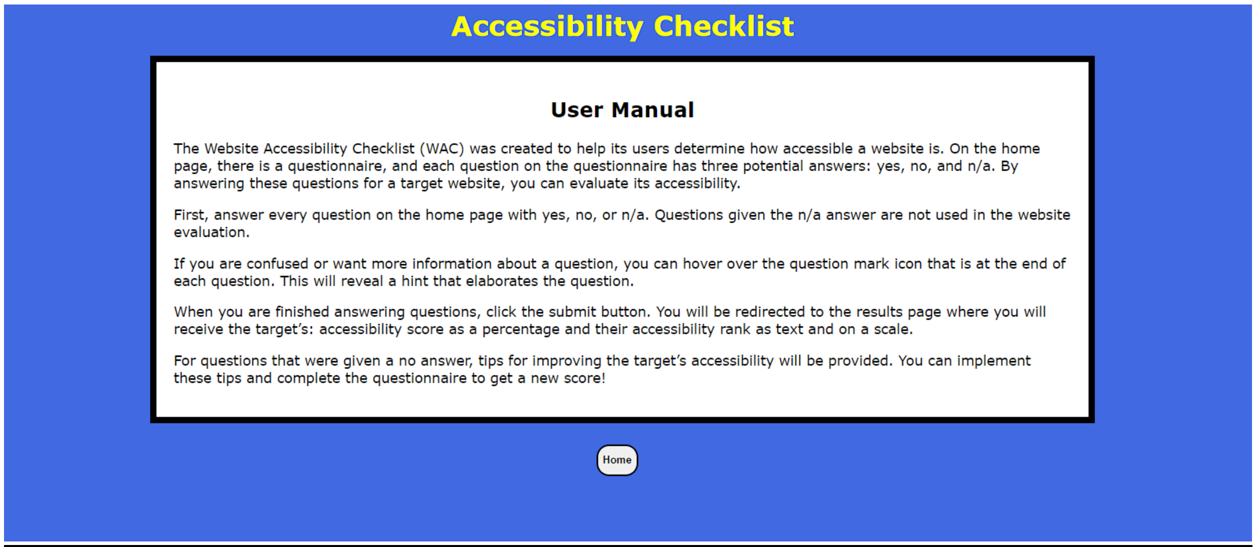
BAD	FAIR	GOOD	GREAT	EXCELLENT

ACCESSIBILITY TIPS

1. Being able to resize helps users who are unable to see well. This helps them read all the information on the screen to their preferences. You are able to implement this feature many different ways. One way to resize is to use the icon with three dots in one column in the top right corner of your screen. If you click on it, there is a section where you can resize the font size. The information needs to stay consistent between the different fonts.

2. A customer service section can be useful for users to acquire more information or clarification. The customer service information contains an email and phone number. Most customer service information is provided at the bottom of the main page and can be accessed from the navigation settings at the top of every page but it's up to you.

User Manual Page



Accessibility Checklist

User Manual

The Website Accessibility Checklist (WAC) was created to help its users determine how accessible a website is. On the home page, there is a questionnaire, and each question on the questionnaire has three potential answers: yes, no, and n/a. By answering these questions for a target website, you can evaluate its accessibility.

First, answer every question on the home page with yes, no, or n/a. Questions given the n/a answer are not used in the website evaluation.

If you are confused or want more information about a question, you can hover over the question mark icon that is at the end of each question. This will reveal a hint that elaborates the question.

When you are finished answering questions, click the submit button. You will be redirected to the results page where you will receive the target's: accessibility score as a percentage and their accessibility rank as text and on a scale.

For questions that were given a no answer, tips for improving the target's accessibility will be provided. You can implement these tips and complete the questionnaire to get a new score!

4.1.2 Objects and Actions

Home Page

- Help Manual Button – Directs the user to the manual page when clicked.
- Application Title – Displays the title of the application.
- Questionnaire Container – Container that holds questions.
- Question – Question in the questionnaire. Each question has the same format consisting of the following elements:
    - Number – Question's place in the questionnaire.
    - Content – Question that the user is asked.
    - Hint Icon – Icon next to the content that presents a textbox that explains the question to the user when held on to or hovered over.
    - Answer Buttons – Radio buttons beneath the content that the user can select to indicate their answer to the question. Only one button can be selected per question.
- Control Buttons – Section beneath the questionnaire that contains buttons that can be used to control the questionnaire. It contains the following buttons:
    - Submit Button – Sends answers to the questionnaire to be processed and directs the user to the result page when clicked.
    - Clear Button – Resets all answers in the questionnaire to their default (i.e. N/A) when clicked.

Results Page

- Help Manual Button – Directs the user to the manual page when clicked.
- Application Title – Displays the title of the application.
- Result Container – Container that holds the results of the questionnaire, including the accessibility score and tip elements.
- Accessibility Score – Percentage calculated using the answers that the user gave to the questionnaire. It is represented as text.
- Accessibility Rating – Textual representation of the score bar.
- Score Bar – Graphical representation of the accessibility score as a progress bar. It consists of the following elements:

- Categories – Equal width sections on the score bar with the following labels going left (low score) to right (high score): bad, fair, good, great, excellent.
    - Progress – Color bar beneath the categories whose width is identical to the accessibility score. It indicates the category that score is in.
- Tips – Section within the result container that contains tips.
    - Tips Title - Displays the title of the tips section within the tips section.
    - Tip – Text which indicates how the user can improve their accessibility score, or provide more “Yes” answers in the questionnaire.
- Home Button – Directs the user to the home page when clicked.

Manual Page

- Application Title – Displays the title of the application.
- Manual Container – Container that holds user manual information.
    - Manual Title – Displays the title of the manual within the container.
    - Manual Content – Content that is presented in the manual, including instructions to assist the user in using the application.
- Home Button – Directs the user to the home page when clicked.

4.2 Interface Design Rules

The user interface has a couple universal design conventions. The first being the font. The UI uses Verdana font for every piece of text. This was chosen because it looks nice and it has good readability. Another almost universal design standard is that text is bolded. The title, questions, buttons, score, and sections are all bold font weight. The only things that don’t use bold text are the question answers and accessibility tips.

When it comes to color, blue was chosen as the whole site background. The same blue is also used for the score bar, and a darker blue was chosen for the question hints. All of the containers and buttons use white as their background color. All text inside the containers and buttons is a regular black, and the borders to containers and buttons are also black. The title was chosen to be yellow just to add a little bit of brightness and variety in color to the website.

4.3 Components Available

The application currently has 4 main UI components available for implementation into the final package. The first is the website design. Design has been mostly completed for the UI, and the basic layout/design of the website has been implemented into a rudimentary and working prototype. The other 3 UI components available are the skeletons for each webpage.

The home page has the containers for the questionnaire setup and the questions designed. The questionnaire still needs to be implemented with the question database to load the questions into the webpage but the design for that is complete. The hints also need to be generated and their functionality implemented. The results page is also complete in design except that it needs to be implemented with the database as well to display accessibility tips. The score processing system also needs to be implemented between these two pages to give the user an accessibility score. The last thing that has been completed is the manual page. The container is all set except help documentation may need to be refined for completeness and altered based on changes to be ready for full release.

4.4 UIDS Description

The prototype user interface presented in this section of the document was developed using visual studio code to edit html/css and the live server extension to run javascript code on it. The UI contains 3 HTML files: 1 for each webpage on the app. It also contains 4 css files: 1 for global use and 1 css file for every page. There are 3 javascript files: 1 for global use, 1 for the home page, and 1 for the results page. Finally there is a JSON file for the questions and related content that will be loaded into the UI.

5.0 Restrictions, Limitations, and Constraints

The development team was restricted on time to complete this design document. The development team had less than a month to determine what tiny tool that they would focus on and the basic functionalities and features needed to help the user determine their application's accessibility. We had to make sure to delegate the workload evenly and complete each section to the best of our ability to create a great tiny tool.

Another constraint is the length of the questionnaire. It should be able to be completed within a ten to fifteen minute interval to maximize the efficiency and effectiveness of the website. If the questionnaire is too long, it could overwhelm the user and could cause the data to be inconclusive. The questions need to be specific and intentional to provide the best analysis of the application and eliminate redundancies. The length of the questionnaire will focus on the qualitative, yes or no, questions.

The website must be able to scale with the user's screen and be able to be viewed and used effectively on mobile devices as well as traditional browsers. All functionality and features need to be available on all screens. The questions must be easy to read, and more information/clarification to each question must be provided to the user in the form of hints, when needed. The navigation of the questions must be clear and separated into different sections based on the type of question. All the questions need to be on the home page and all the answers and tips need to be on the result page.

The site must conduct accurate calculations quickly, and the results page must load within 5 seconds of the user selecting the 'submit' button. The calculations need to be correct and provide a good assessment of the user's application. The weight distribution for the yes or no questions need to also represent the importance to the overall application. If the weight distribution is incorrect, the application in question could be evaluated incorrectly and provide the user with faulty information. These constraints are also outlined in the Performance Bounds (Section 6.3).

6.0 Testing Issues

6.1 Classes of Tests

For more detailed information regarding testing strategy, procedure, and classification, see the test plan for this system.

Testing will be conducted in two ways. Black box testing will have defined inputs and outputs, which will be defined below in Section 6.2. It will be divided into three sections: functional testing, non-functional testing (for improvements on efficiency, user experience, and other features), and regression testing. Functional and non-functional testing will be conducted after each step of coding, and will be repeated as necessary with regression testing as changes are made. For the questionnaire, yes and no questions will be tested to ensure that the correct points are tabulated.

White box testing will test the inner functionality of the code, specifically the calculations that result in the final accessibility score and the weighted table. This is something that we cannot test exclusively with black box testing because black-box testing only provides testers with access to inputs and outputs. This will be done by setting test flags at key points within the code to check each step of the calculation and checking that all the weights are working correctly. Similarly, regression testing will be performed here as well to ensure that subsequent changes are not impacting previous code.

6.2 Expected Software Response

For more detailed test cases, see the test plan for this system.

Black Box		
ID#	**Test**	**Expected Results**
B1	Webpage loads in browser	Webpage loads within five seconds.
B2	Questions added to the database are displayed in the UI	Questions are added to the database and updated in the UI.
B3	‘Yes’ responses are totaled	The final score reflects the responses provided by the

		user.
B4	Tips to ‘no’ responses displayed	Relevant tips are displayed below the final result.
B5	‘?’ provides a hint when clicked	When the user selects the button next to the question, a corresponding hit about the purpose of the question shows.
B6	No score or tip is provided when the user selects ‘NA’	No changes to the output for assessments that do not apply to the application being tested by the user.
B7	All user responses are displayed on the results page.	View all questions and answers
B8	Final result will be displayed	Correct total of all answers is displayed
B9	Progress bar with quality description is displayed	Correct results will be applied based on a conversion from raw score.
B10	View user manual	When the user clicks the button, they are able to navigate to the user manual page
B11	Return Home	Anywhere in the webpage, the user is able to click the home button and they return to the original questions screen.
B12	Navigate to results page	Once the user selects to submit their responses they are automatically sent to the results page.

White Box			
ID#	**Test**	**Execution**	**Expected Results**
W1	Question Score Assignment	After each question, print out the new score to verify the correct sum.	If ‘yes’, then the score is added to the total application score.
W2	Time Metric Score Total	As the user traverses the website and adds time to their total, the new total is printed.	The appropriate amount of time is added for each action as the user navigates the website.

6.3 Performance Bounds

1. **Loading Time** – Browser must load the page within 5 seconds.
2. **Result Delay** – Time between submitting the questionnaire and displaying the

results must be less than 5 seconds.

3. **Question Number** – Expected test metrics should be met as long as the number of questions in the questionnaire is less than or equal to 25.
4. **Average Time** – Average time spent answering questions on the questionnaire must be less than 15 minutes.

6.4 Identification of Critical Components

Critical components of this program will all stem from the host server. Within the actual program itself, collecting and displaying the questions from the database with their respective hints. These questions are the center of the program and not only must they function, but they must be formulated to highlight key components of the application they are assessing. For maintainability, the program should be able to update the current questions and add new ones when necessary.

7.0 Appendices

7.1 Requirements Traceability Matrix

Requirements Traceability Matrix					
Project Name: Website Accessibility Checklist (WAC)					
Business Requirements Document (BRD)		**Functional Requirements Document (FSD)**		**Testing Document**	
BR ID#	**Business Requirement**	**FR ID#**	**Functional Requirement**	**Priority**	**Test Case ID#**
BR_1	Navigation	FR_1	Webpage loads in browser	High	B1
		FR_2	User can read hints to supplement the question information	Medium	B5
		FR_3	User can read user manual	Medium	B10
		FR_4	User can return home from anywhere on the website	Medium	B11
		FR_5	Software automatically navigates to the results page when users submit	High	B12
BR_2	Questions	FR_8	Questions manually added to the database are displayed in the UI	High	B2
		FR_9	Yes responses are totaled	High	B3 W1
		FR_10	No responses are saved and tips for each are displayed on the results page	High	B4 W1
		FR_11	User can input qualitative data into the chart	High	W2
BR_3	Results	FR_13	No score or tip when user selects NA	High	B6
		FR_14	All user responses are displayed	High	B7 B12 W1 W2

		FR_15	Final result is displayed correctly	High	B8 W1 W2
		FR_16	Progress bar with quality description is displayed	Low	B9

7.2 Packaging and Installation Issues

For the packaging of this software, the development team decided to use a zip file to contain the software. A zip file is easy to use and can hold all the software that creates this tiny tool. The zip file can be sent through email, canvas or on a flash drive, making this packaging portable. It shouldn't hinder the usability of the software and should help the user quickly access the software.

The installation should be straightforward using a zip file. The user just needs to unzip the file and have a web browser to access this questionnaire. To unzip the file, the user would either have to open it from their email or canvas or insert a flash drive into their computer. The questionnaire should have no problems, while running after installation.

7.3 Design Metrics to be Used

To evaluate your design we will use a few different design analysis metrics. This is actually very similar to the website we are designing, and we will be using similar strategies ourselves.

1. *System Usability Scale (SUS)* - a ten question survey to be given to users at the end which will gauge the initial usability.
2. *Time on Task* - this will measure the average amount of time that users spend on the website.
3. *Task Accomplishment Rate* - industry standards is around a seventy-five percent task completion rate, but for a site as simple as ours the rate should be as close to 100% as possible.

4. *Error Rate* - this counts the number of errors that the user makes, which will help us mitigate them and evaluate the quality of the end product.

7.4 Supplementary Information

"10 Most Common Web Accessibility Issues to Solve For." BrowserStack, 2 Sept. 2022, https://www.browserstack.com/guide/common-web-accessibility-issues.

Hamilton, Thomas. "White Box Testing – What Is, Techniques" Guru99, 15 Sept. 2022, https://www.guru99.com/white-box-testing.html.

"Introducing JSON." JSON, https://www.json.org/json-en.html.

Kumar, Nishant. "How the Model View Controller Architecture Works – MVC Explained." FreeCodeCamp.org, FreeCodeCamp.org, 30 July 2021, https://www.freecodecamp.org/news/model-view-architecture/.

Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile, https://www.w3.org/TR/mobile-accessibility-mapping/.

Mobile Web Application Best Practices, https://www.w3.org/TR/mwabp/.

"MVC - MDN Web Docs Glossary: Definitions of Web-Related Terms: MDN." MDN Web Docs Glossary: Definitions of Web-Related Terms | MDN, https://developer.mozilla.org/en-US/docs/Glossary/MVC.

Pressman, Roger S. *Software Engineering: A Practitioner's Approach 9th ed*. MCGRAW-HILL COMPANIES (OH), 2005.

"Understanding WCAG 2.1." W3C, https://www.w3.org/WAI/WCAG21/Understanding/.

"What Is Black Box Testing: Techniques" Learning Center, 24 Sept. 2020, https://www.imperva.com/learn/application-security/black-box-testing/#:~:text=Black%20box%20testing%20can%20test,log%20in%20using%20wrong%20credentials.