

Software Requirements Specification

for

C-Teaching-Website

Version 1.0

Prepared by

Group Name: codeTeachers

|  |  |  |
| --- | --- | --- |
| Christian Galvez | 11590801 | christian.galvez@wsu.edu |
| Ethan Pongon | 11639010 | ethan.pongon@wsu.edu |
| <name> | <student #> | <e-mail> |
| <name> | <student #> | <e-mail> |
| <name> | <student #> | <e-mail> |

|  |  |
| --- | --- |
|  |  |
| Date: | <place the date of submission here> |
|  |  |
|  |  |
|  |  |

Content

Revisions iii

1 Introduction 1

1.1 Document Purpose 1

1.2 Product Scope 1

1.3 Intended Audience and Document Overview 1

1.4 Definitions, Acronyms and Abbreviations 1

1.5 Document Conventions 1

1.6 References and Acknowledgments 2

2 Overall Description 3

2.1 Product Perspective 3

2.2 Product Functionality 3

2.3 Users and Characteristics 3

2.4 Operating Environment 3

2.5 Design and Implementation Constraints 4

2.6 User Documentation 4

2.7 Assumptions and Dependencies 4

3 Specific Requirements 5

3.1 External Interface Requirements 5

3.2 Functional Requirements 6

3.3 Behavior Requirements 6

4 Other Non-functional Requirements 7

4.1 Performance Requirements 7

4.2 Safety and Security Requirements 7

4.3 Software Quality Attributes 7

5 Other Requirements 8

Appendix A – Data Dictionary 9

Appendix B - Group Log 10

Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| Draft Type and Number | Full Name | Information about the revision. This table does not need to be filled in whenever a document is touched, only when the version is being upgraded. | 00/00/00 |

# *<In this template you will find text bounded by the “<>” symbols. This text appears in italics and is intended to provide explanations and guide you through the document. There are two types of comments in this document. The comments that are in black are intended specifically for the course. The comments that are in blue are more general and apply to any SRS. Please make sure to delete all of the comments before submitting the documen**t.**>*

# Introduction

Our project is a website that is designed to allow users to learn about C programming and practice writing C code. In this section you will find details about the purpose of this SRS document, a brief description of the product being created and what it hopes to accomplish, and a description of who this document is intended for. In addition to that, there is also information about the conventions and jargon that this document will employ as well as a list of references for items referenced by this document.

*<TO DO: Please provide a brief introduction to your project and a brief overview of what the reader will find in this section.>*

## Document Purpose

This SRS document will cover the C-Teaching-Website v1.0 system as a whole. This document will contain all of the system requirements pertaining to the C-Teaching-Website, as well as detailing user requirements and external interface systems. The C-Teaching-Website v1.0 will be created by the codeTeachers group, who will also be responsible for this document.

<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.

TO DO: Write 1-2 paragraphs describing the purpose of this document as explained above.>

## Product Scope

The C-Teaching-Website final product will be a website that will provide users with an environment to learn about the C programming language and practice implementing its concepts. This usability will be delivered through tutorials and coding challenges that users will work through in a sequential manner. Users will create an account and then start the coding challenges which will involve learning about a C programming concept and then implementing that concept with correct behavior and syntax. To implement the concepts studied in the lessons, users will be able to write and run code directly on the website. Once a user has correctly implemented the concept from the lesson they are working on they will be allowed to continue to the next lesson. The goal of the product is to reduce the roadblocks someone may encounter while trying to learn C programming, for example having to learn how to compile and run code from the terminal. Another benefit of the product is that users will have access to information about concepts they’re learning on the same page they’re writing code on, making it easier to review their code as they write it.

<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals.

TO DO: 1-2 paragraphs describing the scope of the product. Make sure to describe the benefits associated with the product.>

## Intended Audience and Document Overview

This document is intended for the professor and any user interested in the C-Teaching-Website project. This document contains the C-Teaching-Website project’s goals, functionality, users and characteristics, operating environment, design, user documentation, and dependencies. This document also contains external interface, functional and behavior requirements. Non-functional requirements are described as well, such as performance requirements, safety requirements and software quality. For the professor, starting with the introduction and working down the document is the suggested method for reading this document. For the user, reading the introduction in section 1 and reading section 2 may be sufficient.

<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers (In your case it would probably be the “client” and the professor). Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>

## Definitions, Acronyms and Abbreviations

**a.out**: Generic name for executables created from the programming lessons.

**C**: A lower level human readable compiled computer programming language.

**C-Teaching-Website v1.0**: C-Teaching-Website is the name of the project this SRS document describes, and the v1.0 describes the version number of the website this document describes.

**encryptor**: A tool created in C used to encrypt local files on the host machine (server).

**Executable**: Machine code generated from feeding a script into a compiler.

**Express**: A Node.js framework used for creating different web applications.

**GCC**: A compiler for the C programming language.

**JavaScript**: Programming language interpreted by web browsers. Used for the backend of the C-Teaching-Website.

**lesson#\_tests.c**: lesson#\_tests.c files are test files written in C to be used for testing user entered code.

**Node.js**: An asynchronous JavaScript runtime used to run the C-Teaching-Website.

**//#B**: A symbol used to split a lesson#\_tests.c file into two strings.

**passchk:** A file of characters starting with the string “1234” followed immediately by the user’s password.

<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.

TO DO: Please provide a list of all abbreviations and acronyms used in this document sorted in alphabetical order.>

## Document Conventions

This document uses IEEE formatting for citing external sources used. Document uses the Arial font in size 11 and is single spaced. This SRS document uses 1” margins. Text in this document is justified. Bold words inside of a paragraph are used when listing a sequence of elements, such as the definitions described in section 1.4. When this SRS document references specific files or programs used by the C-Teaching-Website, all proper names will be capitalized as defined by that program.

<In general this document follows the IEEE formatting requirements. Use Arial font size 11, or 12 throughout the document for text. Use italics for comments. Document text should be single spaced and maintain the 1” margins found in this template. For Section and Subsection titles please follow the template.

TO DO: Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. Sometimes, it is useful to divide this section to several sections, e.g., Formatting Conventions, Naming Conventions, etc.>

## References and Acknowledgments

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document.

TO DO: Use the standard IEEE citation guide (attached) for this section.> IEEE

# Overall Description

## Product Perspective

The website being produced that this SRS document pertains to is a new, self-contained product. This product was made in response to the lack of interactive and beginner friendly C tutorials online. It is easier for beginners to start writing C code on the website rather than somewhere else because they do not have to worry about compiling and running their code. Instead of managing the compiling and running of code, users enter their code into a section of the website and simply click the run button.

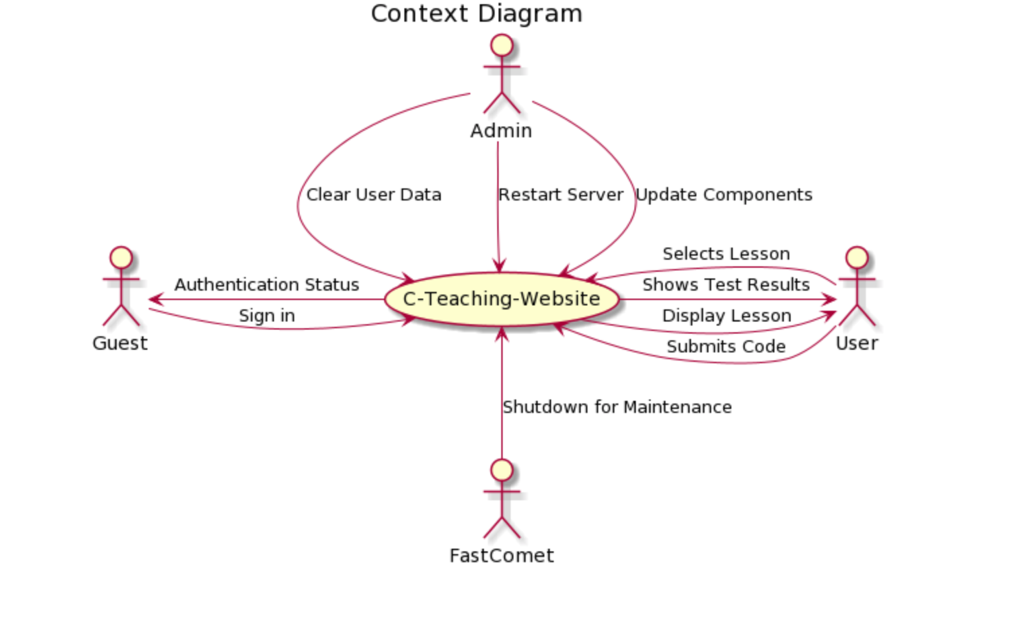


Figure 1: Context Diagram of C-Teaching-Website

<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. In this part, make sure to include a simple diagram that shows the major components of the overall system, subsystem interconnections, and external interface. In this section it is crucial that you will be creative and provide as much information as possible.

TO DO: Provide at least one paragraph describing product perspective. Provide a general diagram that will illustrate how your product interacts with the environment and in what context it is being used, i.e., context diagram.>

## Product Functionality

* Allow users to access a landing page
* Allow users to access an about page
* Allow users to access a login page
* Allow users to create an account
* Allow users to login to their account
* Allow users to read tutorials
* Allow users to write code on the website
* Allow users to run their code on the website
* Provide feedback to users about their code after they run it
* Allow users to access the next tutorial after completing their current one
* Notify users when they have completed the course
* Allow users to view a page that shows their progress in completing the course
* Store user data about their progress and code
* Encrypt user passwords for storage
* Decrypt user passwords for login attempts
* Compile and run C code from users

<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, will be effective.

TO DO:

1. Provide a bulleted list of all the major functions of the system

2. **(Optional)** Provide a Data Flow Diagram of the system to show how these functions relate to each other. This is useful when there is a clear sequence for the functions being performed.>

## Users and Characteristics

There are three main user types that are expected to use this website. The first will be the professor who will be grading the project that this website is being created for. This user is expected to have a large amount of technical expertise as well as complete knowledge of all the concepts being taught through the tutorials on the website. They are expected to use every feature of the website one time at minimum but are not expected to frequently use them. The second expected user type is a user who is new to programming and is using the website to learn. There will be varying degrees of technical expertise with these users but all of them are expected to be very new to the concepts being taught on the website. These users are also the ones who are expected to work on the tutorials the most frequently and to use the account progress tracking feature the most. A third type of user that is expected is one who is moderately experienced with the concepts being taught who also has a larger amount of technical expertise than the users who are very new to the concepts. This type of user is not expected to frequently use the tutorials and will likely not use the progress tracking feature. The two most important users are the professor and the users who are learning programming through the tutorials on the website as they have the most stake in the quality of the website. Additionally, all users will have the same privilege levels on their accounts.

<Identify the various users that you anticipate will use this product. Users may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience.

TO DO:

1. Describe the pertinent characteristics of each user. Certain requirements may pertain only to certain users.

3. Distinguish the most important users for this product from those who are less important to satisfy.>

## Operating Environment

The software will be hosted on a platform that uses Ubuntu version 20.04 as its operating system and uses hardware from the server hosting company “FastComet”. Specifications of the hardware include 15 GB of storage space, 2 CPU cores, 2 GB of RAM and the platform will allow up to 25,000 unique visits to the website per month. The website uses NodeJS to serve its content and this will be implemented through the hosting company’s specialized NodeJS settings panel. Additionally, the website will make use of the GNU Compiler Collection, which is included with Ubuntu, to compile C code.

<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist. In this part, make sure to include a simple diagram that shows the major components of the overall system, subsystem interconnections, and external interface

TO DO: As stated above, in at least one paragraph, describe the environment your system will have to operate in. Make sure to include the minimum platform requirements for your system. >

## Design and Implementation Constraints

There are no limitations regarding timing requirements but there are memory requirements. The hardware will allow up to 15 GB of storage space and user accounts are stored locally on the hosting hardware so there will eventually be a maximum number of users, but this number of users is much higher than how many are expected. Additionally, the Node server will interface with GCC and is restricted to only being able to interact with it through standard input and standard output. Similarly, the user data is being stored in text files which means that when it is accessed it must be done by parsing the file, instead of being able to access specific elements immediately like in other database systems. To keep user data safe, all passwords are encrypted before being stored which means every time a login is attempted there must be a decryption process. Also, every time an account is created there must be an encryption process. We, the creators of the website, are also responsible for the maintaining of the website, so we must check up on it regularly.

<Describe any items or issues that will limit the options available to the developers. These might include: hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer’s organization will be responsible for maintaining the delivered software).

TO DO: In this section you need to consider all of the information you gathered so far, analyze it and correctly identify relevant constraints.>

## User Documentation

The landing page for the website contains brief instructions on how to use the C-Teaching-Website. The instructions describe what user input should look like for each lesson, and what the results page means after submitting a test. The instructions given on the landing page should be sufficient for the user to understand how to use the website fully.

<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.

TO DO: You will not actually develop any user-manuals, but you need to describe what kind of manuals and what kind of help is needed for the software you will be developing. One paragraph should be sufficient for this section.>

## Assumptions and Dependencies

It is assumed that the commercial hosting platform will allow any NodeJS packages as they have support for NodeJS websites. If this was found to not be the case, then it would affect creating cookies and parsing data sent from the frontend of the website to the backend. The website is dependent on both NodeJS as well as GCC to be able to serve content to users and run the code they write. Another assumption is that there will not be a large user base for the website. There is only 15 GB of storage on the hosting platform and user data is stored there so if too many users create accounts then eventually no one else will be able to make a new account. A third assumption is that the website will remain online if the commercial host is not performing maintenance on their servers. If this is not the case, then the website may come offline unexpectedly. A final assumption is that all users are accessing the website from a nonmobile device so they have a large enough screen to view/access all of the content.

<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project.

TO DO: Provide a short list of some major assumptions that might significantly affect your design. For example, you can assume that your client will have 1, 2 or at most 50 Automated Banking Machines. Every number has a significant effect on the design of your system. >

# Specific Requirements

## External Interface Requirements

### User Interfaces

Every page on the C-Teaching-Website will make use of shared CSS elements that create a unified theme throughout the website. The landing page will contain the sidebar, which will be present on every page, as well as the header containing the name of the website and slogan. Additionally, there will be a brief description of the website, and a button that can be clicked to sign in/register. The sign in page will contain the same header and sidebar as the landing page in addition to two fields for creating an account and two for signing in. The sign in page will contain a create account button for the account creation fields, and sign in button for the sign in fields. Once either sign in or create account has been clicked, the user will either be informed that sign in/account creation failed, or the user will be taken to the tutorial pages.

Tutorial pages will have the same sidebar as the landing and sign in pages. Each tutorial will have a similar in style header to the landing page, but the text on the header will describe the current lesson. On every tutorial page, the page will be split into two parts: the tutorial content and the code submission half. The tutorial content, which will include text and visual elements, will be placed on the upper half of the lesson page. The code submission will be placed on the lower half of each lesson page. The code submission area of the page will have a box for typing in code with a submit button next to it. After a user submits their code, they will be brought to a test results page that grades the users’ code by correctness. Test results page will use same styling as lesson page.

When the user clicks the progress button on the sidebar they will be taken to a page with the same styling as the landing page. The new page will display a graphic that indicates how many tutorials a user has completed, and how many they have remaining. Similarly, when the user clicks the about button on the sidebar, they are taken to a new page with the same styling as the landing page. This page will contain text talking about the purpose of the website.

*Graphical user interface, text, application, email

Description automatically generated*

Figure 2: Example of Website Uniform Styling (Sidebar and Header)

<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., Cancel) that will appear on every screen, error message display standards, and so on. Define the software components for which a user interface is needed.

TO DO: The least you can do for this section is to describe in words the different User Interfaces and the different screens that will be available to the user. Optional: You may also provide an initial Graphical User Interface design (does not have to be final).>

### Hardware Interfaces

**Mouse/Touchscreen:** Users are expected to access the website with a mouse or touchscreen for website navigation.

**Keyboard:** Users are expected to access the website with a keyboard, so the user can input text into text fields.

**User Device:** The user’s device must have a screen large enough to display the website content correctly. The user’s device must also have a modern web browser.

**NodeJS:** NodeJS will communicate with the user by serving page content, as well as accepting and processing input. NodeJS will access the server’s GCC compiler, as well as run executables on the server.

<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware. You are not required to specify what protocols you will be using to communicate with the hardware, but it will be usually included in this part as well.

TO DO: Please provide a short description of the different hardware interfaces. If you will be using some special libraries to communicate with your software mention them here. In case you have more than one hardware interface divide this section into subsections.>

### Software Interfaces

The host server will be running Ubuntu version 20.04 and will be running the NodeJS server v10.19.0. The NodeJS server will create and monitor child processes on the host, which will make use of the stdin and stdout features of Ubuntu. Some of the child processes will make use of the GCC 9.3.0 C compiler to compile user code.

<Describe the connections between this product and other specific software components (name and version), including databases, operating systems (Windows? Linux? Etc…), tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.

TO DO: The previous part illustrates some of the information you would usually include in this part of the SRS document. To make things simpler, you are only required to describe the specific interface with the operating system.>

### Communications Interfaces

When the user creates an account, their password is stored on the host server in a text file that is encrypted using an XOR cipher. XOR cipher is used when a user attempts to login to their account. One current issue with the XOR cipher is that when logging in it must be used to decrypt followed by an encrypt to make sure user data is not corrupted. The corruption may occur if a user logs in and the server suddenly crashes at that moment.

The server runs all child processes and opens all files synchronously to ensure all logic is completed in the intended order. If processes and I/O were not run synchronously it could crash the server. If files were opened asynchronously then it is possible that another user could access someone else’s data.

The host server serves website pages to users using HTTP.

<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.

TO DO: Do not go into too much detail, but provide 1-2 paragraphs were you will outline the major communication standards. For example, if you decide to use encryption there is no need to specify the exact encryption standards, but rather, specify the fact that the data will be encrypted and name what standards you consider using. >

## Functional Requirements

**Website Pages:**

When the user visits the website URL in their browser, they will be brought to a landing page.

When the user clicks on the “About” button on the sidebar, the website will load the about page.

When the user clicks “Get Started” on the landing page, the website will load the login page.

When the user clicks the “Progress” tab on the sidebar, the website will load a progress page visually depicting the user’s progress in the course.

**User Accounts:**

If the user does not have an account, the website will allow them to create one when given a new username and password.

If the user enters a username on the login page and the user exists in the system, the website will attempt to login.

If a user inputs a username that exists and gives the correct password, the website will keep that user logged into their account.

If a user inputs a username that exists and gives an incorrect password, the website will prevent that user from logging in.

After a user successfully logs into their account, the user should be brought to the lessons page.

**Lessons:**

When a user clicks on a lesson with their mouse, they will be brought to that corresponding lesson page.

When a lesson page is opened, the user will be able to read tutorials.

If the lesson page is opened, the user will be able to enter code in a field directly on the website.

When the user has entered code on the lesson page and clicked the submit button, the website will compile and test the entered code.

When the user submits code for a lesson, the website will load a results page that grades the entered code using tests.

When the user loads the results page, the results page will display the tests that passed and failed.

When the user completes a lesson by passing all the tests, the website will grant access to the next lesson.

When the user has completed all available lessons, the website will notify the user that they have completed the course.

When the user opens a previously completed lesson, their previously entered code should be present in the code entry field.

**User Data:**

When the user submits code on a lesson page, the website will locally store the submitted code in the user’s directory.

When a user progresses through a lesson, the website will store data about the user’s progress.

When a login attempt is made by a guest, the website will decrypt the passchk file and re-encrypt the file after the attempt has been made.

When a guest creates an account successfully, the website will write the password to a file called passchk and encrypt it with the password.

*< Functional requirements capture the intended behavior of the system. This behavior may be expressed as services, tasks or functions the system is required to perform. This section is the direct continuation of section 2.2 where you have specified the general functional requirements. Here, you should list in detail the different product functions with specific explanations regarding every function.*

*TO DO: Break the functional requirements to several functional areas and divide this section into subsections accordingly. Provide a detailed list of all product operations related to these functional areas.>*

## Behavior Requirements

### Use Case View

Diagram

Description automatically generated

Figure 3: Use Case Diagram for C-Teaching-Website

**Actors:**

User – The user visiting the website to learn C using the lessons.

Admin – Administrators who have access to the backend of the C-Teaching-Website.

FastComet – Commercial hosting service for the C-Teaching-Website.

**Use Cases:**

Load Landing Page – Loads the landing page in the user’s browser.

Load About Page – Loads the about page in the user’s browser.

Load Sign in Page – Loads the sign in page in the user’s browser.

Create Account – Creates a new user directory on the server.

Account Exists – If the user folder already exists for the user’s username, give the user an error message and do not login.

Sign in – Attempts to login with a username and password entered by the user.

Incorrect Username or Password – If the username and password do not match on the sign in page, give the user an error message and do not login.

Load Progress Page – Loads a page in the user’s browser visually depicting their progress throughout the online course.

Load Lesson – When the user selects a lesson on the progress page, a new lesson page is loaded in the user’s browser for that corresponding lesson.

Submit Code – The user will enter code into a field and click a submit button.

Save Code – The server will grab the code entered by the user and save it in their respective user directory.

Compile Code – The code submitted by the user will be compiled by GCC.

Run Tests – The user’s compiled code is tested, and the results are returned to the server.

Load Results Page – Using the results from Run Tests, the results page will display which tests passed and failed.

<A use case defines a goal-oriented set of interactions between external actors and the system under consideration.

TO DO: Provide a use case diagram which shows the entire system and all possible actors. Do not include detailed use case descriptions (these will be needed when you will be working on the Test Plan), but make sure to include a short description of what every use-case is, who are the actors in your diagram.>

# Other Non-functional Requirements

## Performance Requirements

1. User submitted code will not take more than 10 seconds to compile and test.
2. Account creation will not take more than 5 seconds.
3. Account sign in will not take more than 10 seconds.
4. Website will allow for more than 1 concurrent user at a time.
5. Website pages will not take more than 10 seconds to load.

<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.

TODO: Provide relevant performance requirements based on the information you collected from the client. For example you can say “1. Any transaction will not take more than 10 seconds, etc…>

## Safety and Security Requirements

**User Password Encryption:**

* The password that is given to the C-Teaching-Website when a user creates an account will be encrypted using an XOR cipher created in C.
* The passchk file in the user’s directory will be unreadable by any actors without the corresponding password.

**User Password Decryption:**

* The user’s password will be decrypted and re-encrypted when the user attempts to login with a given password.
* The server will not shutdown during a user login. In the case that a shutdown occurs during a user login, the user’s website folder will not become corrupted as a whole, but the passchk file can become corrupted. If this occurs, the user will need to contact an administrator to repair their account.

Due to the use of the XOR cipher for encrypting user passwords, user passwords will be protected against all actors who may interact with the system backend directly.

<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product’s design or use. Define any safety certifications that must be satisfied. Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements.

TODO:

* Provide relevant safety requirements based on your interview with the client or, on your expectation for the product.
* Describe briefly what level of security is expected from this product by your client and provide a bulleted (or numbered) list of the major security requirements.>

## Software Quality Attributes

<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.

TODO: Use subsections (e.g., 4.3.1 Reliability, 4.3.2 Portability, etc…) provide requirements related to the different software quality attributes. Base the information you include in these subsections on the material you have learned in the class. Make sure, that you do not just write “This software shall be maintainable…” Indicate how you plan to achieve it, etc.>

# Other Requirements

<This section is **Optional.** Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A – Data Dictionary

**Constants**

|  |  |
| --- | --- |
| **Name** | **Definition** |
| express | Contains the “express” library |
| app | Contains the return value of the express() call which starts the hosting of the website |
| cookieParser | Contains the “cookie-parser” library |
| bodyParser | Contains the “body-parser” library |
| info | Contains the “console” library |
| urlencodedParser | Contains the “urlencoded” portion of the bodyParser variable |
| port | Contains the environment’s specified port or if there is not specified environment it contains the port 8081 |
| spawn | Contains a function from a the “child\_process” library that allows NodeJS to create and continuously manage a separate process on the host |
| exec | Contains a function from the “child\_process” library that allows NodeJS to call an executable or command |
| execSync | Contains a function from a the “child\_process” library that allows NodeJS to call an executable or command synchronously |
| fs | Contains the library that deals with the file system on the host |
| server | Contains the return value of the call to express to listen on a port |
| encryptorPath | Contains the file path to the directory containing the encryption software |
| usersPath | Contains the file path to the directory containing the user profiles |

**Variables**

|  |  |
| --- | --- |
| maxErrors | Holds the number of errors present in the current C test. |
| errorSum | Holds the sum of the number of errors present in the current test. |
| fp | Holds a pointer to a file struct for reading text files in C. |
| cur | Current character in second argument (password) for encryptor tool. |
| length | Length of the password passed through to the encryptor. |
| password | Password passed through to the encryptor tool as a command line argument. |
| curr | First character from the passchk file. |
| pIndex | Index of character used for XOR cipher in the encryptor tool. |
| UserAccount.username | Contains the username passed to UserAccount object |
| UserAccount.password | Contains the password passed to the UserAccount object |
| key | Contains the value 1234 |
| user | Contains the value of UserAccount.username from the object this method is called on |
| pass | Contains the value of UserAccount.password from the object this method is called on |
| key | Contains the value of UserAccount.key from the object this method is called on |
| attempt | Contains true or false based on whether the attemptLogin() succeeds or fails |
| data | Contains the string of text that represents a users encrypted password |

**Inputs and Outputs**

|  |  |  |
| --- | --- | --- |
| **Methods** | **Input** | **Output** |
| app.get() | / | Sends the homepage html file |
| app.get() | /sidebar.css | Sends the css file containing the sidebar styling |
| app.get() | /home.css | Sends the css file containing the homepage styling |
| app.get() | /About | Sends the About page html file |
| app.post() | /go | Sends the login page html file |
| app.post() | /login | Logs user in and sends html file for first coding tutorial |
| app.post() | /createacc | Creates new account for user and sends html for file for first coding tutorial |
| userExists() | No input | Returns true or false based on whether or not the object’s defined user exists in the user database |
| existsSync() | Filename | Synchronously checks if a file with the name passed in exists and returns true or false based on if it does or not |
| createUser() | No input | Return true if a user was able to be created with the object’s defined user and returns false if it was unable to create a user |
| userExists() | No input | Returns true if the object’s defined user exists in the database already and false otherwise |
| mkdirSync() | usersPath + this.username | Returns nothing but creates a directory with the name of the argument passed in |
| writeFileSync() | Path to specified user’s encrypted password, this.key + this.password | Returns nothing but writes a file that contains a users encrypted password |
| exec() | Path to specified executable or command, command line arguments | Returns nothing but runs executables or commands |
| attemptLogin() | No input | Returns true if successfully logged in or returns false if not |
| execSync() | Path to specified user’s encrypted password location | No output but re-encrypts user’s password |
| readFileSync() | Path to specified file, file encoding type | Returns string object of all file text |
| sendFile() | Path to specified file | No output but sends file to user on website |

*<Data dictionary is used to track all the different variables, states and functional requirements that you described in your document. Make sure to include the complete list of all constants, state variables (and their possible states), inputs and outputs in a table. In the table, include the description of these items as well as all related operations and requirements.>*

Appendix B - Group Log

October 13, 2020, Meeting 1:

Time 120 minutes

Ethan: Proof of concept. Node has a thing called child processes which allow you to execute commands from node basically. There is an input field, and you can click submit. Takes the text, puts it in main.c, and the child process runs the execute command. Does that and compiles the code the user gave it through the website. There is another child process called spawn, and it has another object that is holding the process. Write all of the stdout from a.out from gcc and put it into a file. For every coding problem they have to do, ask for really specific output and compare to the desired output. The thing we're looking at right now is just looking for a minimum viable product. Highlight correct code in green, incorrect in red. Was thinking for this project it will probably be okay to just not actually micromanage how they are doing the output. If people want to name their variables something else, we would have to do a lot of string parsing. Thinking we would either have people return one of their variables, which isn't super intuitive when you're just starting out in C. We would have to make it so they would take arguments from a command line.

How we could handle each lesson: We have tests that are run in the backend. We place user code inside of functions that will have tests run on it. We could highlight errors from the compiler describing the common error. When you press run the code, it needs to send a new page to send the output.

Would be okay if we do something where we just have a standardized page that we go to for the results of each problem. You write the code and it will take you to the next page. The page where we compare tests would have to be on a separate page from where you write your code.

The website will be hosted on fastcomet.

Christian is going to work on the C tests, Ethan is going to work on the landing page.

Look up "run C program with node child processes" for info on compiling C.

Deliverables by next meeting:

Ethan: -Create the landing page and sidebar that will be on every page

-Write information for 1.0 and 1.2 on SRS document

Christian: -Create the unit tests for the first C challenge

-Write information for 1.1 and 1.3 on SRS document

Backlog:

-Set up page for user account creation/sign in

-Set up user account system

-Set up node function for running C code

-Error highlighting for C tests

-Compilation failed page

-Error handling for the compilation and running of C code (gcc failing, race conditions, etc.)

-Set up About page

October 20, 2020, Meeting 2:

Time 120 minutes

Christian and Ethan went over changes made. Main landing page with CSS style sheets were created, with the first lesson and the backend for the first lesson.

Christian:

* SRS 1.4, 1.5, 2.6
* Set up user account system
* Implementing encryptor tool for user passwords

Ethan:

* SRS 2.3, 2.4, 2.5
* Set up page for user account creation/sign in
* Set up About page

Backlog:

-Set up page for user account creation/sign in

-Set up user account system

-Specific C tests that passed/failed

-Compilation failed page

-Set up About page

-UI for code and lesson on challenge pages

-Implementing encryptor tool for user passwords

October 28, 2020:

Time 100 minutes

We went over the User account system, About Page, Login Page.

Christian:

* UI for code and lesson on challenge pages
* Cookie parser for cookie header

Ethan:

* cookie checking on all pages that work with user accounts
* hook up user account system to login/sign up page

Backlog:

-Specific C tests that passed/failed

-add content to homepage about how the tutorials work

-Compilation failed page

<Please include here all the minutes from your group meetings, your group activities, and any other relevant information that will assist the Teaching Assistant to determine the effort put forth to produce this document>