

Software Requirements Specification

for

C-Teaching-Website

Version 2.0

Prepared by

Group Name: codeTeachers

|  |  |  |
| --- | --- | --- |
| Christian Galvez | 11590801 | christian.galvez@wsu.edu |
| Ethan Pongon | 11639010 | ethan.pongon@wsu.edu |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |
| --- | --- |
|  |  |
| Date: | December 16, 2020 |
|  |  |
|  |  |
|  |  |

Table of Contents

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 2

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 4

2.4 Operating Environment 4

2.5 Design and Implementation Constraints 4

2.6 User Documentation 4

2.7 Assumptions and Dependencies 5

3 Specific Requirements 6

3.1 External Interface Requirements 6

3.2 Functional Requirements 7

3.3 Behavior Requirements 8

4 Other Non-functional Requirements 10

4.1 Performance Requirements 10

4.2 Safety and Security Requirements 10

4.3 Software Quality Attributes 10

5 Other Requirements 12

Appendix A – Data Dictionary 13

Appendix B - Group Log 16

Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
| --- | --- | --- | --- |
| V1 | Ethan Pongon  Christian Galvez | Initial version of the SRS document completed. | 11/6/2020 |
| V2 | Ethan Pongon  Christian Galvez | \* Fixed alphabetical ordering in Section 1.4 Definitions list.  \* Moved C, Express, GCC, JavaScript, Node.js definitions to Section 2.7 Assumptions and Dependancies.  \* Renamed a.out in Section 1.4 to program due to the change in the software.  \* Reconstructed Section 2.3 to better outline the different user types that are expected to use the website.  \* Removed NodeJS from Section 3.1.2 Hardware Interfaces and removed spacing from the list.  \* Split Section 1.2 into two different paragraphs and added additional information.  \* Reformatted lists in Section 3.3.1 to have consistent styling with other lists in SRS.  \* Specified User actor as Professor, Beginner and Intermediate in Section 3.3.1.  \* Changed list in Section 4.1 from numerical to bulleted.  \* Revised Section 1.5 Document Conventions to include undefined list formats.  \* Changed date on document to December 16, 2020.  \* Updated group log as of 12/10/2020.  \* Updated Table of Contents page numbers to match with each section. | 12/??/2020 |

# 

# 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.

## 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.

## Product Scope

The C-Teaching-Website 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 website is to streamline the process for new users wanting to learn the C programming language. The user will simply need to log into the website and follow the tutorials. This removes the obstacle of needing to first learn command line tools to compile and run code. Due to the lesson content and code entry existing on the same page, the user can read tutorials as they type code. This creates a comfortable and painless environment for beginners to learn how to code in C.

## 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.

## Definitions, Acronyms and Abbreviations

**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.

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

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

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

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

## 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. Undefined lists as present in Section 2.2 will use bullet points for each statement. 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. IEEE citations will follow the IEEE citation guide found on IEEE-DataPort [1]. Software requirements will be worded using Chris Rupp’s template [2].

## References and Acknowledgments

[1] A. Outman, "How to Cite References: IEEE Documentation Style," IEEE-DataPort, Help & Support. [Online]. Available: https://ieee-dataport.org/help/how-cite-references-ieee-documentation-style. [Accessed: Nov. 6, 2020].

[2] C. Rupp, "Requirements Templates -- The Blueprint of your Requirement," SOPHIST GmbH, Webinhalte zu Kapitel 10, 2014. [Online]. Available: https://www.sophist.de/fileadmin/user\_upload/Bilder\_zu\_Seiten/Publikationen/RE6/Webinhalte\_Buchteil\_3/Requirements\_Templates\_-\_The\_Blue\_Print\_of\_your\_Requirements\_Rupp.pdf. [Accessed: Nov. 6, 2020].

# 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.

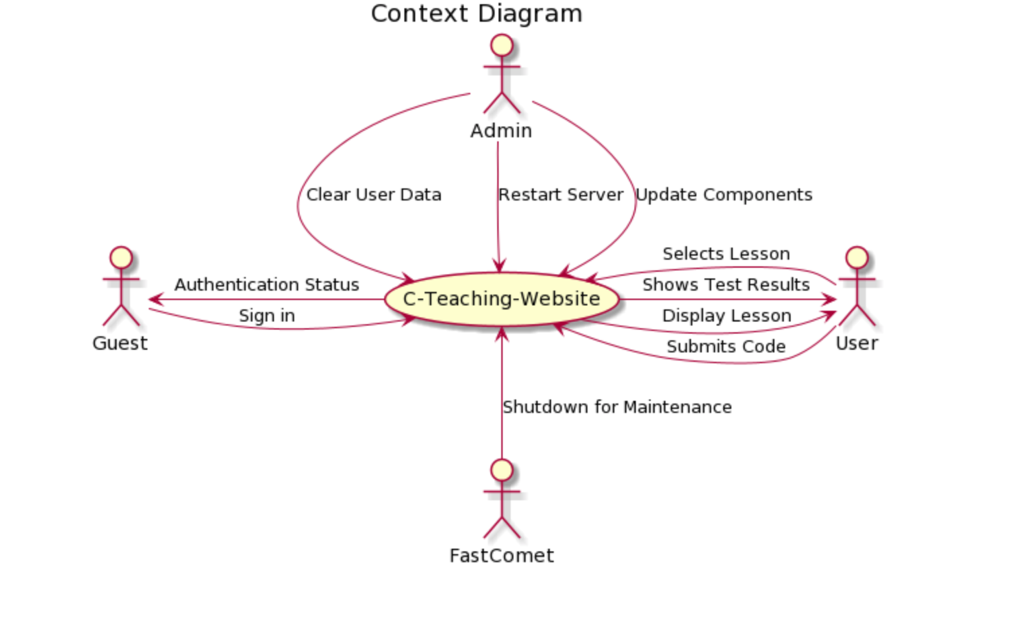


Figure 1: Context Diagram of C-Teaching-Website

## 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

## Users and Characteristics

Three main types of users are expected to use the website:

* Professor: Has the most technical knowledge about the subject being taught through the website. Expected to use every feature on the website at least once, but not expected to use the features with frequency.
* Beginner: Has the least amount of technical knowledge about the subject being taught through the website. Expected to stress the tutorials the most and check user progress frequently.
* Intermediate: Has a moderate amount of technical knowledge about the subject being taught through the website. Expected to not frequently use tutorials, and not expected to use the progress tracking feature.

The two most important users are the professor and the beginners, as they have the most stake in the quality of the website. The professor, beginner and intermediate user types will all have the same privilege levels on their accounts.

## 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.

## 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.

## 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.

## Assumptions and Dependencies

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

**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.

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

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.

# 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)

### 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.

### 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.

### 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.

## 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.

## 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. Contains Professor, Beginner and Intermediate user types.
* 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.

# Other Non-functional Requirements

## Performance Requirements

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

## 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.

## Software Quality Attributes

* + 1. **Availability**

The website will be available to access at any time of day on any given day, except when fastcomet is performing server maintenance.

* + 1. **Reliability**

The website will never fail to load page content or run and score user code.

* + 1. **Correctness**

Tutorials present on the website will all be correct and follow proper C syntax and accepted C programming practices.

* + 1. **Reusability**

The website tutorials are built according to the same layout and reuse layout assets to make it easier for the developers to create more tutorials.

* + 1. **Adaptability**

Assets for the website can easily be retooled to add another set of tutorials to teach a programming language other than C. Additionally, text content on the website can easily be translated by a browser for non-English speakers.

* + 1. **Portability**

The executable tools utilized by the website come with a makefile which will allow them to be recompiled to fit another operating system if there was a change in server hosting.

* + 1. **Usability**

The website will display all elements correctly on all monitors 800x600 pixels and larger.

# Other Requirements

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 |

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, Meeting 3:

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

November 18, 2020, Meeting 4:

120 min meeting

Ethan allowed for the sign in page to save a cookie for the user when the log in and take them to the home page when logged in successfully. Christian added the styling to the lesson 1 page. Sections 1.0, 1.2, 1.3 and 2.1 were added to the software design document.

Christian:

* Specific C tests that passed/failed - Add theme to results page.
* Create more lessons.

Ethan:

* Create progress page.
* Implement user progress tracking.
* Add content to homepage about how the tutorials work.
* Set up FastComet hosting for website.

Backlog:

* Link all of the pages together.
* BONUS: Set up express routes for dynamic HTML loading.
* Add progress tracking to each tutorial page once they are on the main branch
* Add "welcome {user}" to green bar once a user has logged in using document.cookie on HTML files

December 8, 2020, Meeting 5:

120 min meeting

Christian created lessons 2 and 3 on the betaLessons branch, added theming to the results page and created more advanced C tests. Completion page was also created. Ethan set up FastComet for website hosting, added more information to the homepage, and progress page.

Christian:

* Implementing user progress tracking into lesson pages - fix /nextlesson refresh bug

Ethan:

* Make lessons clickable from progress page
* Make Lesson 4
* Make Lesson 5

Backlog: