

CS 320 Course Project Final Report

for

C-Teaching-Website

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

Contents ii

1 Introduction 1

1.1 Project Overview 1

1.2 Definitions, Acronyms and Abbreviations 1

1.3 References and Acknowledgments 1

2 Design 2

2.1 System Modeling 2

2.2 Interface Design 2

3 Implementation 7

3.1 Development Environment 7

3.2 Task Distribution 7

3.3 Challenges 8

4 Testing 9

4.1 Testing Plan 9

4.2 Tests for Functional Requirements 9

4.3 Tests for Non-functional Requirements 10

4.4 Hardware and Software Requirements 10

5 Analysis 11

6 Conclusion 12

Appendix A - Group Log 13

# Introduction

Our project is a website that is designed to allow users to learn about C programming and practice writing C code. The goal of our website is to allow beginners to get a head start in C programming. The website was designed with beginners in mind, so the content of the lessons is aimed solely at newcomers.

## Project Overview

The C-Teaching-Website is a website where users can learn C by following several coding tutorials. Each tutorial on the website contains a field with lesson content, and a field for code entry. Right next to the code entry box is a submit button that allows the user to submit code to the server to be compiled and tested. After submitting code, the user is presented with a results screen showing the quality of their entered code. There are a total of 5 lessons on the website, each one building off of concepts from the last.

## Definitions, Acronyms and Abbreviations

**Chai:** JavaScript library used for making assertions in unit testing.

**FastComet:** Commercial hosting service for the C-Teaching-Website.

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

**Mocha:** JavaScript library used for automated unit testing.

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

**npm:** Node package manager. Used for installing packages and dependencies for NodeJS projects. Automated tests in this project utilize the npm test command.

**X86\_64:** Intel 64-bit architecture.

## 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].

# Design

## System Modeling

*Diagram

Description automatically generated*

Figure 1: Revised C-Teaching Website Class Diagram

Our class diagram was updated to feature new variables and methods created for each class. Other than the class diagram, our implementation strictly follows the design document (milestone 2).

## Interface Design

*Text, letter

Description automatically generated*

Figure 2: Landing Page of Website for New Users

*A picture containing text, screenshot, indoor

Description automatically generated*

Figure 3: Landing Page of Website for Returning Users

Text

Description automatically generated

Figure 4: About Page

*Graphical user interface, text, application, email

Description automatically generated*

Figure 5: Lessons Page

*Graphical user interface, text, application, email

Description automatically generated*

Figure 6: Lesson Results Page (Compiled Correctly)

Graphical user interface, text, application

Description automatically generated

Figure 7: Completion Page

Graphical user interface, text, application, email

Description automatically generated

Figure 8: Lessons Results Page (Did Not Compile Correctly)

Graphical user interface

Description automatically generated

Figure 9: User Progress Page

# Implementation

## Development Environment

**IDEs Used:**

* IntelliJ Ultimate Edition

**Text Editors User:**

* Atom
* Sublime Text
* Vim
* Nano
* VSCode

**Git Software:**

* Sublime Merge
* Git
* GitHub

**Linting Software:**

* ESLint

**Testing Software:**

* Mocha
* Chai

**Programming Languages:**

* JavaScript
* C
* HTML
* CSS

**Misc:**

* Makefile
* cs320-code-style.xml

## Task Distribution

Tasks for this project were assigned during scrum meetings, where group members were assigned to finish a set of tasks in a given time period. Each week the backlog of tasks for group members to complete would grow depending on how the development of the project changed over time. In total, each group member completed each of these tasks:

**Christian Galvez:**

* Create the unit tests for the first C challenge
* Write information for 1.1 and 1.3 on SRS document
* SRS 1.4, 1.5, 2.6
* Set up user account system
* Implementing encryptor tool for user passwords
* UI for code and lesson on challenge pages
* Cookie parser for cookie header
* Specific C tests that passed/failed - Add theme to results page
* Create more lessons
* Implementing user progress tracking into lesson pages - fix /nextlesson refresh bug

**Ethan Pongon:**

* Create the landing page and sidebar that will be on every page
* Write information for 1.0 and 1.2 on SRS document
* SRS 2.3, 2.4, 2.5
* Set up page for user account creation/sign in
* Set up About page
* Cookie checking on all pages that work with user accounts
* Hook up user account system to login/sign up page
* Create progress page
* Implement user progress tracking
* Add content to homepage about how the tutorials work
* Set up FastComet hosting for website
* Make Lesson 4
* Make Lesson 5

## Challenges

Most challenges in developing the website were caused by race conditions, due to many tasks being done asynchronously. These issues were solved by either changing these processes to run synchronously or using observers on processes to perform actions after they completed.

While the website is running it constantly reads and writes to files on the backend, which created many bugs due to corrupted or missing files. These issues were solved by creating new edge cases in the code that handled missing or corrupted files.

# Testing

## Testing Plan

**Classes/Methods:**

* CookieCipher
* UserAccount
* ResultsPage

**Functions:**

* getProgress
* findProgress
* createCFile
* failedTests
* updateProgress
* getFailedDesc

**Website Routes:**

* /Progress
* /go
* /prev
* /login
* /creatacc
* /submission

**External Processes:**

* GCC exec calls
* program spawnSync calls
* encryptor execSync calls

**Testing Plan:**

* The CookieCipher and UserAccount classes will have unit tests written for them using mocha and chai by December 12, 2020.
* getProgress, createCFile, failedTests, and updateProgress will all have unit tests written for them by December 12, 2020.
* The findProgress, getFailedDesc, ResultsPage, all website routes, and external processes were tested during the entire Testing Phase when the project was available to the public. From December 12, 2020, to December 15, 2020 the website will be tested with concurrent users.

## Tests for Functional Requirements

Test for correctness

UserAccounts testing

✓ Create user when username available

✓ Do not allow user creation when username taken

✓ Log into account with correct password

✓ Deny account login with wrong password

CookieCipher testing

✓ CookieCipher correctly parses cookie header string

Progress tracking correctness

✓ getProgress returns 1 for new accounts

✓ updateProgress changes user progress

createCFile creates proper C file

✓ createCFile properly generates lesson 3 file

failedTests correctly returns an array for incorrect tests

✓ failedTests(255) returns [1, 2, 3, 4, 5, 6, 7, 8]

✓ failedTests(0) returns []

✓ failedTests(42) returns [2, 4, 6]

## Tests for Non-functional Requirements

Tests for performance

Account creation will not take longer than 5 seconds

✓ Creating Basic User Account (8ms)

✓ Creating Advanced User Account (7ms)

Account sign in will not take more than 10 seconds

✓ Signing into a User Account (21ms)

## Hardware and Software Requirements

**Hardware Requirements for Tests:**

* x86\_64 PC
* Keyboard
* Monitor
* Internet connection

**Software Requirements for Tests:**

* GitHub repository for C-Teaching-Website must be cloned onto the PC.
* Node and npm must be installed.
* Mocha and chai must be installed globally.

# Analysis

**Christian Galvez:**

* Milestone 0 – 1 hour.
* Milestone 1 – 30 hours.
  + Milestone 1 took the most effort because during this milestone our group had to plan exactly how the website will work and what it will look like. This milestone also included the work on the SRS document, which was the longest document out of them all.
* Milestone 2 – 11 hours.
* Milestone 3 – 20 hours.

**Ethan Pongon:**

* Milestone 0 – 1 hour.
* Milestone 1 – 25 hours.
  + Milestone 1 took the most effort because it was the milestone with the SRS report, which was by far the longest document. Additionally, during this milestone is when most of the website’s code was planned and implemented.
* Milestone 2 – 8 hours.
* Milestone 3 – 17 hours.

# Conclusion

During this project we learned a lot of lessons about managing a large software system. The first of those lessons was getting accustomed to writing reusable code. At a couple points in the project, we realized we had rewritten very similar code in several places when we could’ve just used one version of the code in a function. After realizing this we went through our code and created functions that could apply in several places to replace the old, similar, blocks of code. Another lesson we learned was always to keep concurrency in mind when it is a requirement of the project you are working on. We ran into several bugs while creating our website that were due to lack of planning around having several concurrent users. The final lesson we took away from this project was how to efficiently divide work in a group. To plan how the project was going to get done we used a modified version of the agile development method we learned in class. Each week we would meet and discuss the state of the project, add any new work to our running backlog and assign new work to ourselves for the next sprint. This was a very effective way of ensuring each component of the project would get completed.

Appendix A - Group Log

codeTeachers met every 1 to 2 weeks, and all group members effectively communicated with one another during meetings. Every group member was fully engaged, and all assigned work was completed, and all future work was planned accordingly.

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 Lesson 4
* Make Lesson 5

Backlog: