

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 |
|  |  |
|  |  |
|  |  |

Content

# 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 3

3.1 Development Environment 3

3.2 Task Distribution 3

3.3 Challenges 3

4 Testing 4

4.1 Testing Plan 4

4.2 Tests for Functional Requirements 4

4.3 Tests for Non-functional Requirements 4

4.4 Hardware and Software Requirements 4

5 Analysis 5

6 Conclusion 6

Appendix A - Group Log 7

# 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.

*<TO DO: Please provide a brief introduction to your project.>*

## 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.

< A brief description of the project.

TO DO: Write 1-2 paragraphs describing the project. >

## 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.

<Define all the terms necessary to properly interpret the report, including acronyms and abbreviations.

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

## 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].

<List any other documents or Web addresses to which this document 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 for this section.>

# 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).

< Update your UML diagrams in milestone 2, to reflect the real implementation of this software.

TO DO: Provide an updated version of the UML diagrams, including use case diagrams, sequence (or state) diagrams, activities diagrams, and class diagrams. If you don’t have an updated version, just mention: “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, email

Description automatically generated

Figure 7: Lessons Results Page (Did Not Compile Correctly)

Graphical user interface

Description automatically generated

Figure 8: User Progress Page

Graphical user interface, text, application

Description automatically generated

Figure 9: Completion Page

<Provide several screenshots to illustrate your interface design.

TO DO:

For each subsystem, pick one or two representative screenshots and paste here.>

# 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

<Describe the development environment you were using for the project.

TO DO: List the programming languages, IDEs, tools, etc.>

## 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

*<Describ how the implementation tasks are distributed among team members.*

*TO DO: For each team member, describe his/her main implementation tasks in this project.*

*If this is a one-person project, mention: “all the work presented here is done by \*\*\* (your name).” >*

## 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.

*<This section is optional. Describe the challenges in the implementation, if there are any, and how you dealt with them.*

*TO DO: If you don’t have anything to fill in, just leave this section blank.>*

# Testing

## <*This section is a summary of your testing report>*

## 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.

<Describe your testing plan for the project.

TODO: Give a list of items or functions you want to test, and also a schedule for performing the testing. >

## 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]

<Describe your test results for the functional requirements.

TODO: Provide a list of use cases or functions you have tested, as well as the testing results (whether or not the system passed the tests).>

## 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)

<Similar to the Section 4.2, but this section is for the non-functional requirements. >

## 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.

<Describe the hardware and software requirements for performing the tests. >

# 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.

<In this Section you need to analyze the effort that has been put on this project.

TODO: Describe how many hours (approximately) each team member spent on the project, for each milestone, which milestone took the most effort and why. >

# 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.

<Conclude the document with what you have learned through working on the project.>

Appendix A - Group Log

< Describe how frequently the group members meet during the semester, and how effective the communication is. This is optional for one-person projects.>