**A2 Computing Coursework**

**Code College**

****

# ­Table of Contents

Contents

[Table of Contents 2](#_Toc449545527)

[Analysis 4](#_Toc449545528)

[Background Problem 4](#_Toc449545529)

[Description of Current System 4](#_Toc449545530)

[Identification of the End User 4](#_Toc449545531)

[Identification of User needs and limitations 4](#_Toc449545532)

[Data Sources and Destinations 5](#_Toc449545533)

[Data Volumes 5](#_Toc449545534)

[Analysis Data Dictionary (what data the User needs) 5](#_Toc449545535)

[Entity Relationship Diagram (ERD) 6](#_Toc449545536)

[Data Flow Diagram (DFD) 7](#_Toc449545537)

[Objectives 7](#_Toc449545538)

[Potential Solutions 7](#_Toc449545539)

[Proposed Solution 9](#_Toc449545540)

[Design 10](#_Toc449545541)

[Overall System Design 10](#_Toc449545542)

[Modular Structure of System 10](#_Toc449545543)

[Data Dictionary 11](#_Toc449545544)

[System Security 12](#_Toc449545545)

[User Interface Design (HCI) 12](#_Toc449545546)

[Implementation 13](#_Toc449545547)

[Screenshots 13](#_Toc449545548)

[Test Plan 15](#_Toc449545549)

[System Maintenance 17](#_Toc449545550)

[System Overview 17](#_Toc449545551)

[Login 19](#_Toc449545552)

[Sign Up 19](#_Toc449545553)

[Dashboard 20](#_Toc449545554)

[Exercise 20](#_Toc449545555)

[Interpreter 21](#_Toc449545556)

[Marker 22](#_Toc449545557)

[AddExercise 24](#_Toc449545558)

[ExerciseParser 24](#_Toc449545559)

[Exercise File 25](#_Toc449545560)

[Appraisal 27](#_Toc449545561)

[Appendix 29](#_Toc449545562)

[Test Evidence 29](#_Toc449545563)

[Code 39](#_Toc449545564)

[App\_Start 44](#_Toc449545565)

[Controllers 46](#_Toc449545566)

[Interpreter 53](#_Toc449545567)

[Marker 162](#_Toc449545568)

[Models 166](#_Toc449545569)

[Views 185](#_Toc449545570)

The User Manual is also included at the end of this document. It can be found after page 208.

# Analysis

## Background Problem

Our nation’s children are not equipped for the 21st century’s dependence on computers. We need programmers more than ever and at the moment, not enough exists.

Teachers in schools are underqualified at teaching code, a tool that can aid teachers to teach code would be invaluable in modern education.

Lua is an easy to learn language useful for its simplicity, portability, and practicality.

## Description of Current System

Currently, the market ruler for programming teaching websites is *Codecademy.* It uses simple tutorials to teach the basics. Unfortunately, it does not teach Lua, it instead teaches languages such as Python, JavaScript, Ruby and others.

There are a few other attempts at creating websites to teach but they are somewhat boring and often complex. Students just zone out.

## Identification of the End User

The end user will be young people and maybe underqualified teachers tasked with teaching programming to students.

My old IT teacher, Mr. Brown, asked me to write a website that would be useful for teaching Lua to his students. He does not know Lua very well and so is relying on the website to do most of the teaching for him. His students are aged between 10 and 16, however, the site can be used by anyone and it is aimed at all ages to use.

Mr. Brown currently uses textbooks to teach his students. These are often very ineffective, they have little to no exercises and aren’t very engaging. It is widely respected that the only true way to learn to program is through practical experience. That is what my project aims to achieve.

## Identification of User needs and limitations

The website needs to be interesting and engaging but also informative and intuitive. Mr. Brown only has a certain amount of time to teach and to set homework per week. The website needs to be able to teach well in a short space of time but also to possibly be interesting enough for the user to want to do more in their own time.

In an interview with Mr. Brown, I asked what programming concepts he felt should be in the end product. He told me that he didn’t want anything too complex, just the basics that beginners should know.

List of Topics That Should Be Covered:

* Printing to the console
* Variables
* Arithmetic
* Conditionals
* Loops

These should be taught through a series of exercises that are concise, intuitive and informative.

## Data Sources and Destinations

|  |  |  |
| --- | --- | --- |
| Data | Source | Destination |
| User Details | User Input | Storage in System |
| User Progress | Calculated by Server | Storage in System |
| Code | User Input | Server Interpreter |
| Feedback on Code | System | Student |
| Track of Task Completed | System | Student |
| Exercise | Admin Input | Storage in System |

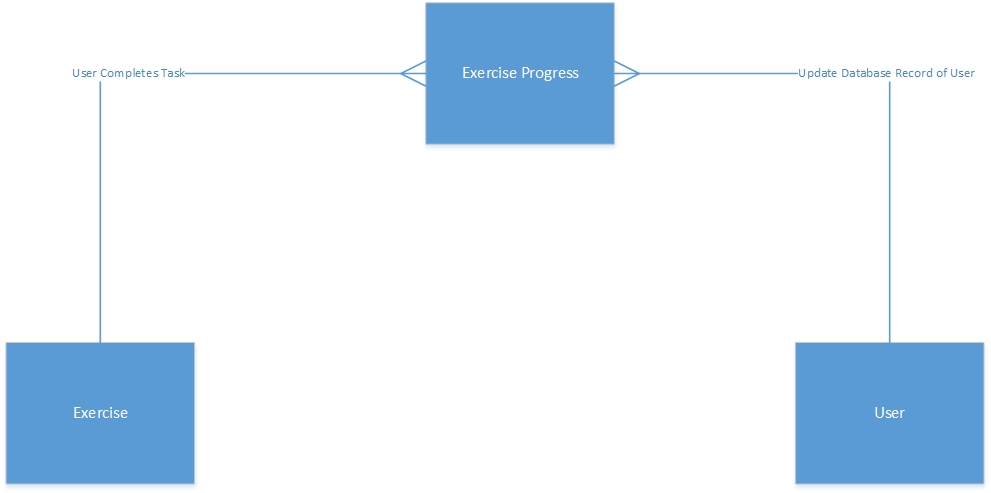
## Data Volumes

The website will be available to anyone to use. However, when just being used by Mr. Brown and his students, about 20 users are expected. My system should be able to handle the load of many users; I would be more limited by server load.

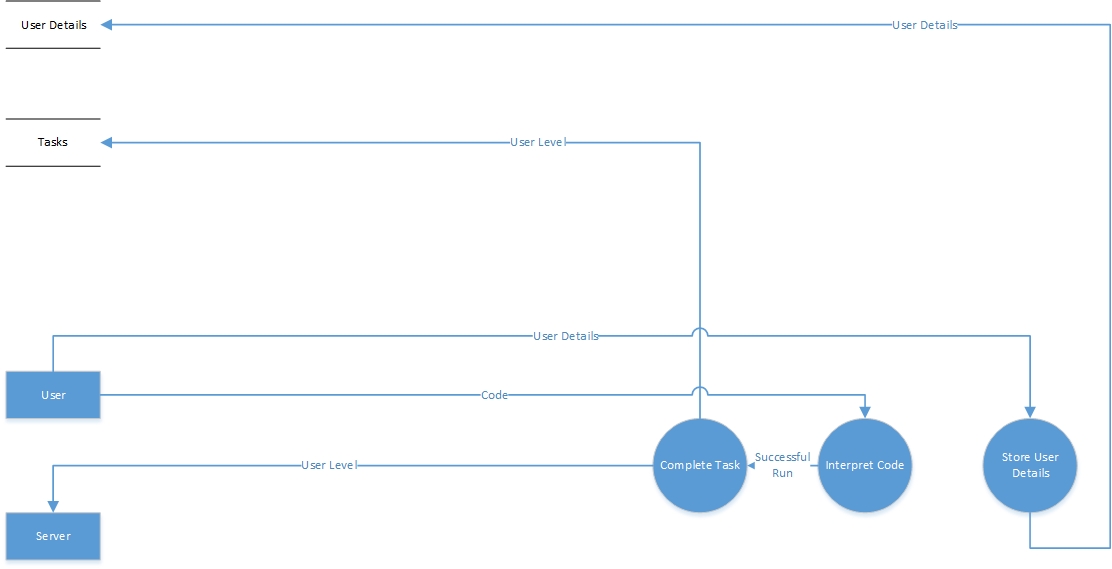
## Analysis Data Dictionary (what data the User needs)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Field | Field Type | Field Size | Field Description | Field Necessity |
| Name | String | 50 | Name of user | Refer to user |
| Email | String | 50 | Email of user | Email and identify user |
| Username | String | 50 | Username of user | Login with and identify user |
| PasswordHash | String | 512 | SHA512 hash of user’s password | To login with |
| UserLevel | Int | 5 | Level of user | Identify where user is in the course |

## Entity Relationship Diagram (ERD)



## Data Flow Diagram (DFD)

****

## Objectives

1. The system should be able to interpret the user’s code and mark it sufficiently.
2. The system should run the user’s code in a window on the browser.
3. The system should store the users’ details in a secure and professional fashion.
4. The system should parse exercises and store them correctly.
5. The system should be secure and be as invulnerable to attack as possible.
6. Communications should be as impervious to attack as possible.
7. Interpreting and marking of code should be done quickly. Ideally, no greater than 55-60ms.

## Potential Solutions

|  |  |  |
| --- | --- | --- |
| Suggested Solutions | Advantages | Disadvantages |
| ASP.NET MVC 5 using Entity Framework (EF) | I have knowledge of C#, .NET and EF.  The .NET Framework has numerous libraries available that allow me to save time and enhance the system. E.G. System.Cryptography, this library contains classes and functions for hashing text. This means I can hash passwords and make the system more secure and less vulnerable to MITM attack of the user.  C# is a fast and easy language that would make it far easier to write an interpreter.  The Entity Framework makes database management a very simple, fast and efficient. | In ASP.NET 4.6, you can only run web applications on a Windows Server using IIS.  IIS can be difficult to setup.  Can only be programmed within a Windows environment. |
| PHP and MySQL | Common practice, this paradigm is common in industry.  Cross-platform development and execution, can be run on Linux, Windows, and Mac OS X. | I have not much practice in using these technologies.  There are not as many tools readily available to you. Heavy use of 3rd party libraries or writing my own time-consuming algorithms is not ideal.  Speed is also important. Writing an interpreter in an interpreted language would slow down user experience. |
| Node.js and MySQL | Cross-platform development and execution, can be run on Linux, Windows, and Mac OS X.  It is my understanding that due to it being a C++ library, you can use C++ wrappers to run C++ code. This would make the interpreter exponentially faster (perhaps, faster than necessary) even if it does make the code more verbose and complex. | I have not much practice in using these technologies.  Verbosity and complexity are possibly critical factors.  The server may be a bit more complex to set up than with other technologies. |

## Proposed Solution

**ASP.NET MVC 5 using Entity Framework (EF)**

I picked these technologies for a few different reasons. Firstly, I am most proficient in .NET and it would save time reading through docs, learning the other options.

Secondly, the vast capabilities and libraries of C# and the .NET Framework makes it ideal for this situation. I will have to spend less time writing lots of code that would not be the first priority. Also, boilerplate code would be less of a problem when you use C# and Visual Studio 2015. These functions will make my code more secure, user-friendly and efficient.

ASP.NET MVC has numerous abilities that are extremely useful in web development. These such functionalities include *Razor*:

*“Razor is a markup syntax that lets you embed server-based code (Visual Basic and C#) into web pages.”*

Server-based code can create dynamic web content on the fly while a web page is written to the browser. When a web page is called, the server executes the server-based code inside the page before it returns the page to the browser. By running on the server, the code can perform complex tasks, like accessing databases.

This tool makes interpreting the code, listing the exercises and dealing with the logged in user much easier. Especially, as all the .NET libraries are readily accessible in Razor.

# Design

## Overall System Design

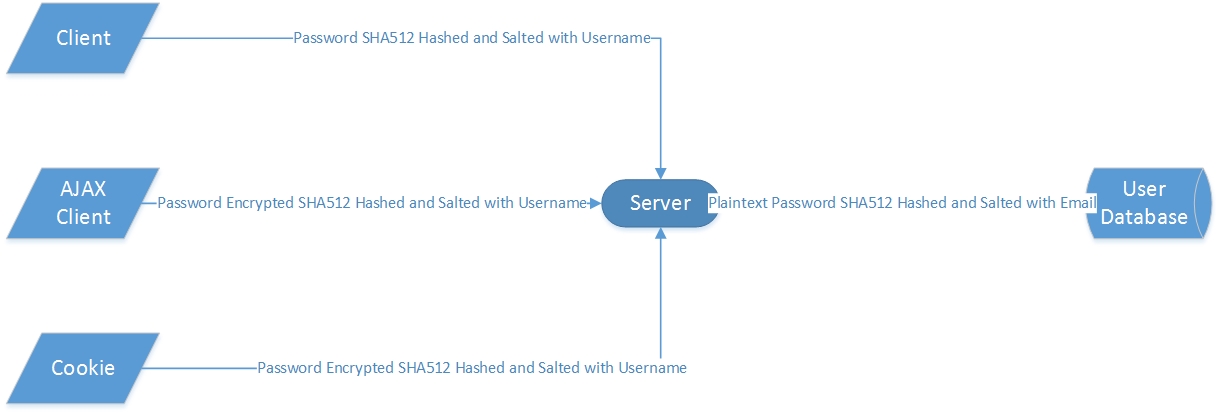
My system will run in IIS on a Windows Server. I will be using ASP.NET 4.6. This uses C# for client-side and server-side processing as well as other web technologies for the front-end. E.G. HTML, CSS and JavaScript. For the database used for storing user data, I am going to be using the Entity Framework. This is a platform for Object-Relational Mapping (ORM) database management in .NET.

|  |  |
| --- | --- |
| **Inputs** e.g.  User Details  User Code Attempt  Exercise Files | **Processing** e.g.  Interpret and Mark Code  Process User Data  Output Code  Parse Exercise |
| **Storage** e.g.  User Details Database  Exercise Database | **Outputs** e.g.  Exercises  Code Output |

## Modular Structure of System

Website Structure:

Security Model:



The above diagram shows the various stages of sensitive data transmission from client to server. The node labeled “Client” represents the traditional web pages that transmit data through HTTP POST e.g Login, SignUp. The node “AJAX Client” represents the AJAX communication between client and server on the exercise page this is done using HTTP POST also. The “Cookie” node represents the HTTP Cookie stored in the browser and the data stored in the Cookie.

## Data Dictionary

User Database:

|  |  |  |  |
| --- | --- | --- | --- |
| Field Name | Data Type | Validation | Reason |
| Name | String | No numbers or special characters | Refer to user |
| Email | String | Incorporates @ and . | Email and identify user |
| Username | String | No special characters | Login with and identify user |
| PasswordHash | String |  | To login securely with |
| UserLevel | Int |  | Identify where user is in the course |

Exercise Database:

|  |  |  |
| --- | --- | --- |
| Field Name | Data Type | Reason |
| DBID | Int | Acts as DB key |
| ExID | Int | Refer to exercise |
| ExTitle | String | Title of exercise |
| ExDescription | String | Description of exercise |
| ExCodeTemplate | String | Code that is put in the editor for the user to change e.g an if statement where the user has to only change the condition |
| ExAppendCode | String | Code to be appended to the user’s code for the interpreter e.g a print statement for set variables |
| ExMarkScheme | ExMarkScheme | Mark Scheme object. Contains information on how to mark the code written by the user for the given task |

## System Security

The user’s password is hashed by the client and by the server. This is to ensure that the system is less impervious to a Man-In-The-Middle (MITM) attack.

When the user’s password is stored in a cookie or transmitted via AJAX, the hashed password is also encrypted, this makes it theoretically impossible to crack.

The web application is hosted on IIS Windows Server. Windows Server has lots of features that make it secure but my server has numerous features to make it more secure. Only port 80 is open, this makes it less vulnerable to an outside attack. It has anti-virus software to prevent from viruses. It is also hosted on a network rigged with Network-Intrusion-Devices (NIDs), Cisco firewalls, and other monitoring/security devices. I also only access the server through the most secure VPN in the world, Apollo Anywhere.

The user’s data is stored in an Entity Framework database stored on the server. It is stored in a secure and Data Protection Act compliant way.

The server is often backed up and in the unlikely event of a crash, it will reboot itself automatically. This minimizes downtime as much as possible.

I am planning the connection to the server for users to be encrypted through SSL or TLS for extra protection from MITM and other forms of attack.

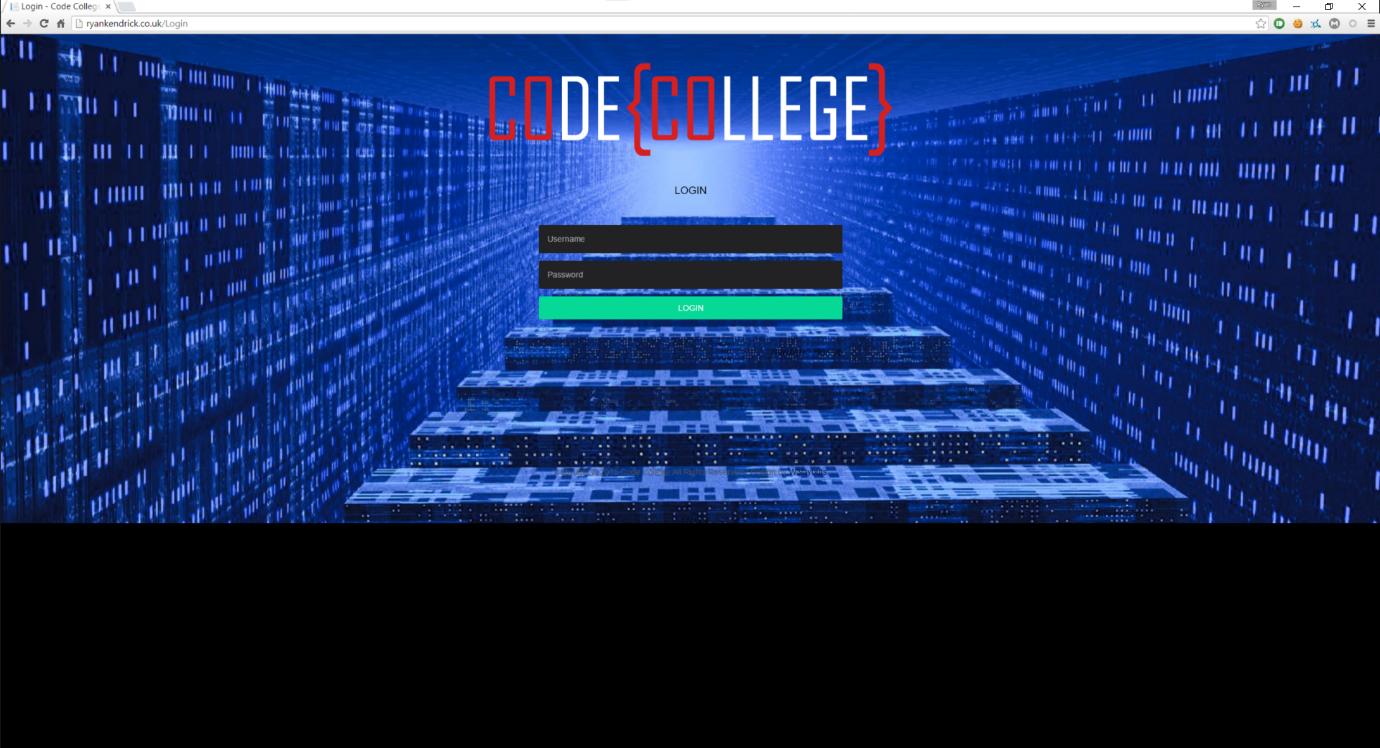
## User Interface Design (HCI)

The design in my project uses a green and black color scheme. This creates a pleasing and easy on the eyes design. The color appeals to a younger audience as well as anyone learning to code. The colors green and black are often associated with code by the media such as computer terminals in movies.

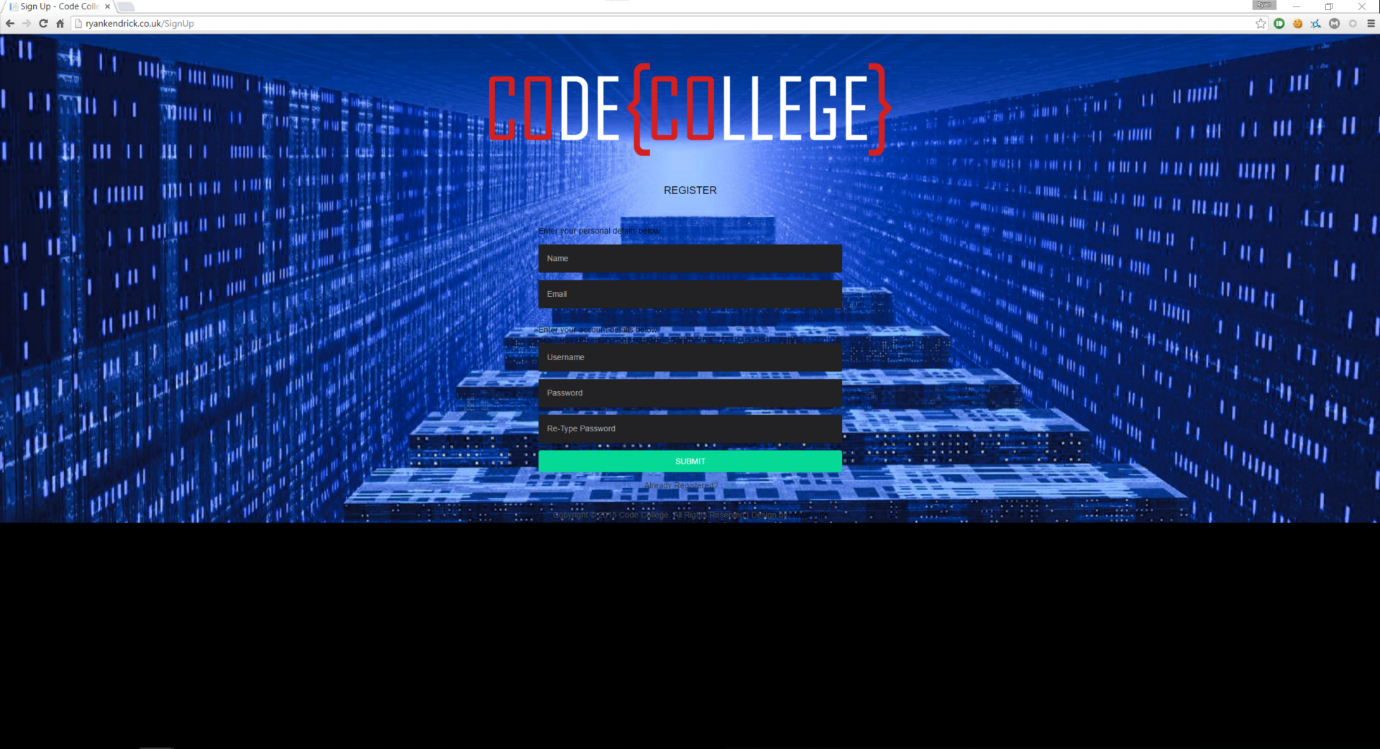
The control structure is also intuitive and easy to understand, perfect for kids.

# Implementation

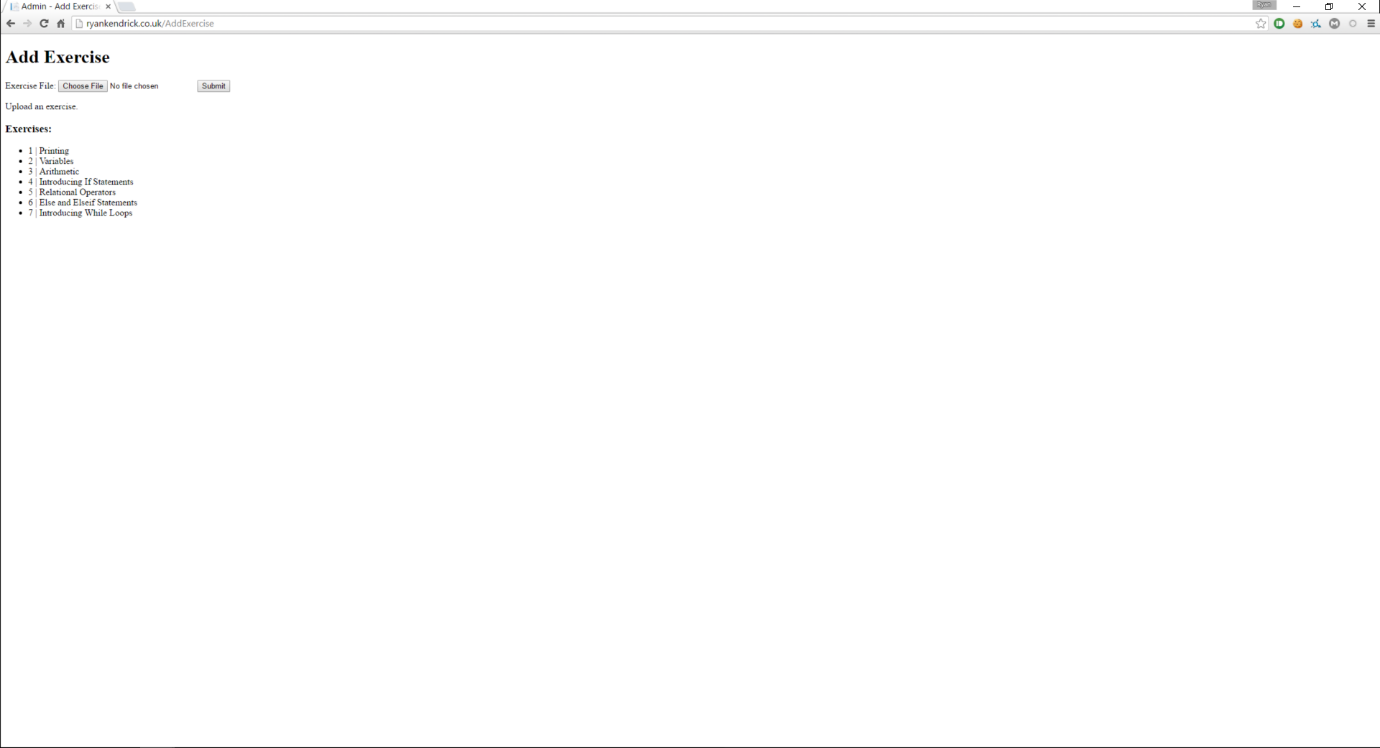
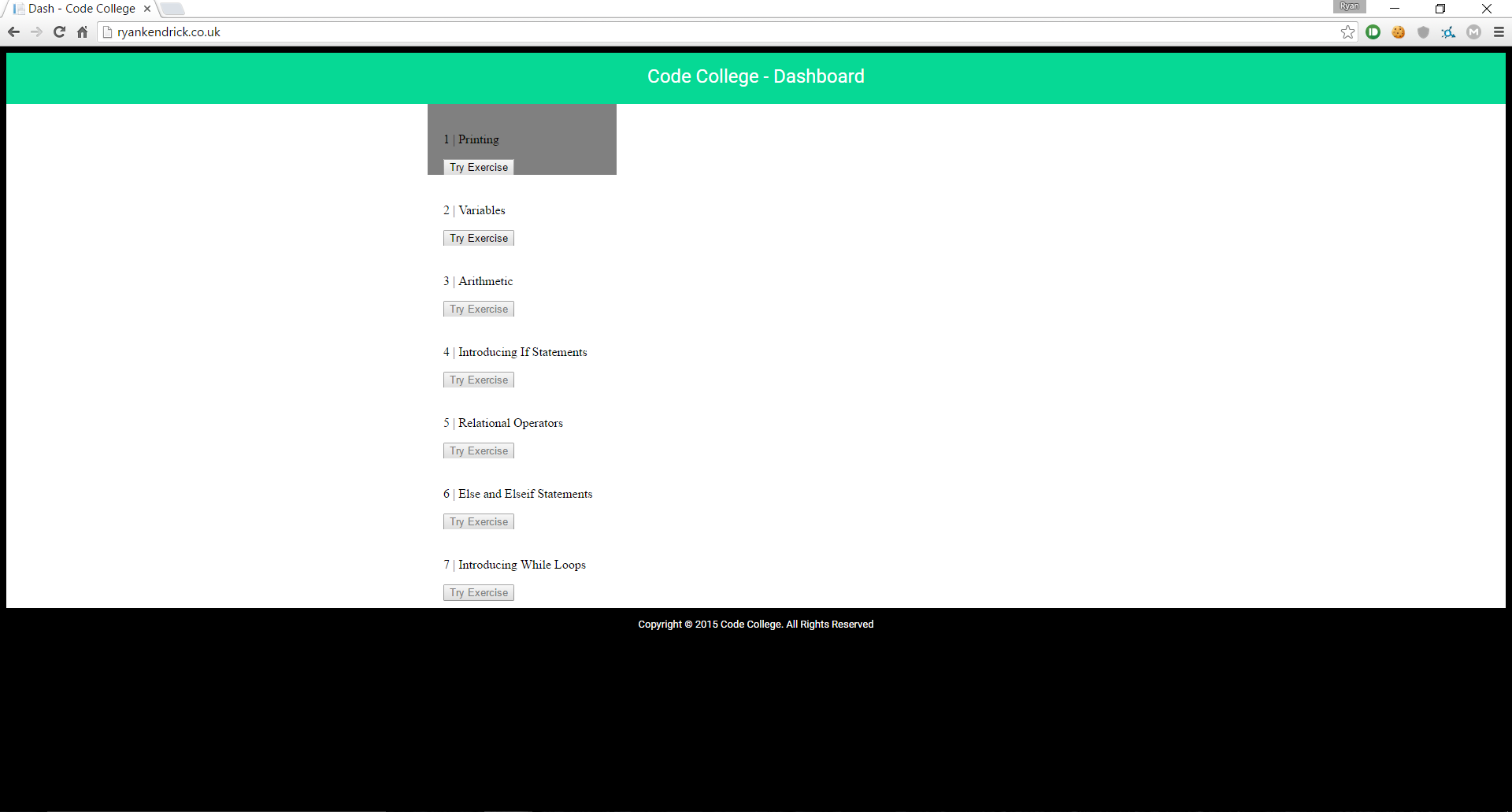
## Screenshots

Login View:

Sign Up View:



Add Exercise View:

Dashboard View:

# Test Plan

Each test has evidence in the ‘Test Evidence’ section of the appendix. They are referenced using the Test Number.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Series and Number | Module(s) Being Tested | Description | Test Data | Expected Result | Actual Result |
| 1.1 | LoginController | On launch, login screen should display | Open website | Login screen displayed |  |
| 1.2 | LoginController, Account, and DashboardController | User logins in | Username = ‘jsmith’ Password = ‘guest123’ | User dashboard displayed |  |
| 1.3 | DashboardController and ExerciseController | Student navigates to an exercise | Button clicked | Exercise page shows with the correct exercise loaded |  |
| 1.4 | Interpreter and ExerciseController | User submits incorrect code | Code = Any code that doesn’t work  Button click | The page to tell the user that they are wrong and show an error message in the console |  |
| 1.5 | Interpreter and ExerciseController | User submits correct code | Code = Any code that works to achieve the exercise goal | A success message to show and the page to navigate to the next exercise |  |
| 1.6 | DashboardController | User’s completed exercises are greyed out on the dashboard | Complete any exercise | All the completed exercises will be blanked out whereas the unfinished exercise is colored |  |
|  |  |  |  |  |  |
| 2.1 | Account and SignUpController | Create new user | Name = ‘Sterling Archer’  Email = ‘archer@gmail.com’  Username = ‘Gator2’  Password = ‘Guest’ | The page diverts to the dash page for the newly created user | As Expected |
| 2.2 | Account, LoginController, and DashboardController | Go to Login with a logged in account cookie and have the website redirect you to the Dash | Cookie for account ‘Gator2’ | The site reads the cookie and redirects you to the Dash of the logged in user | As Expected |
| 2.3 | ExerciseController and LoginController | Go to Exercise without logging in and have the site redirect you to the Login |  | The Login screen should appear | As Expected |
|  |  |  |  |  |  |
| 3.1 | AddExerciseController | Upload a new exercise to the AddExercise page and have it added to the database | New exercise | The file should be uploaded and the new exercise added | As Expected |
| 3.2 | AddExerciseController | Upload an invalid exercise file and have it be rejected | Test.txt | The file should be rejected by the server and nothing should be added | As Expected |
|  |  |  |  |  |  |
| 4.1 | Interpreter | Submit code and have the code be passed to the interpreter and have the interpreter create the global environment |  | The same code written in the browser passed to interpreter and LuaTable object created | As Expected |
| 4.2 | Interpreter | Submit code and have Chunk be created with no errors. |  | The Chunk object for the code created with no errors | As Expected |

# System Maintenance

## System Overview

This system is an ASP.NET MVC 5 web application written in C# 6 using Entity Framework 6 and the .NET Framework 4.6.

The main parts of the code are as follows:

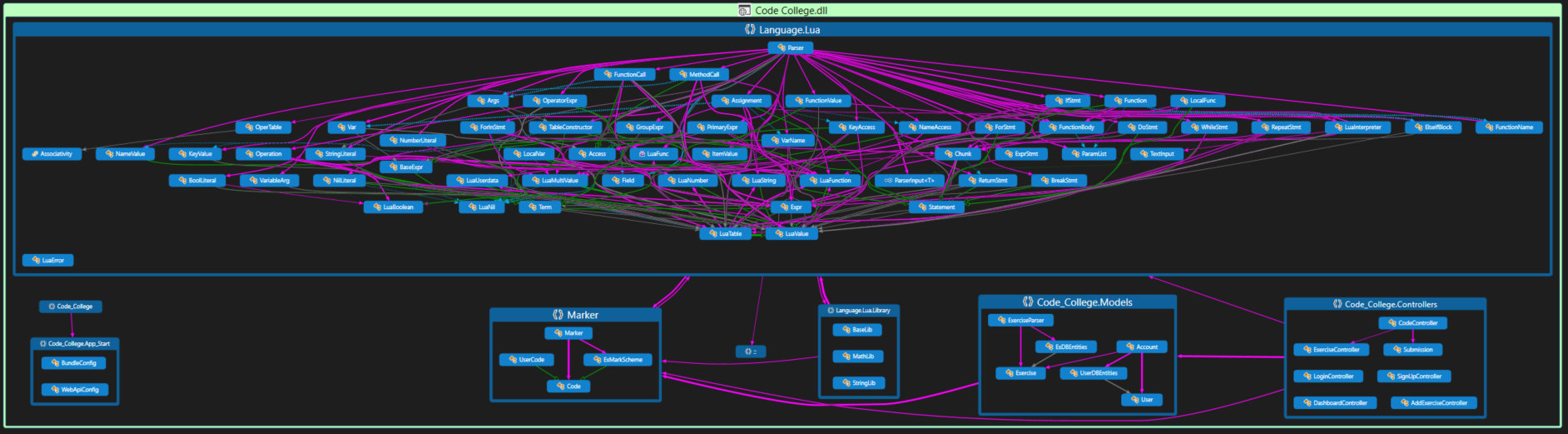
* Controllers: These handle the back-end code for the views. However, CodeController is a controller that deals with WebAPI which is what the AJAX talks to.
* Interpreter: This is the biggest part of the project; it handles all areas of interpretation of the code.
* Marker: This handles the marking of the code after it has been interpreted.
* Models: These are backend components that handle specific tasks such the database context, exercise parsing and account methods.
* Views: These are the Razor pages that are processed on the server and rendered by the browser. The C# in these files are run by the server but the HTML, CSS and JavaScript are all run and rendered on the user’s browser.

Other areas of the project are:

* Content: Contains external CSS files.
* Scripts: Contains external JavaScript files.
* App\_Start: Contains important configuration code that runs at startup.

There are two databases involved in this system. The first is the user database, this stores all the account data for users. This is called UserDBEntities. The second is the exercise database, this stores all the data for the exercises, including the mark scheme. This is called ExDBEntities.

On the next page is a full object diagram of the project, it also links what inherits and references what.



## Login

When a user navigates to any view apart from the AddExercise view on the site, the site checks to see if you have a valid cookie. A valid cookie indicates that the user has already logged in.

When you navigate to this page, if you are already logged in, you will be redirected to your dashboard.

The Login page consists of two text entries and a submit button. When you hit the submit button, the username and password that the user put in the respective textboxes are transmitted to the server via HTTP POST.

The server stores the username in a string object called Username. It then salts and hashes the password with an SHA512 hash and salts it with the username.

During the verification process, the server will create a user object and will check the database to see if it can assign it to a user from the DB that has the same username. If a user with the same username is found, the server hashes the password again using an SHA512 hash but this time salts it with the email of the user object that was just assigned from the database. This newly hashed password is checked against the password of the user object. If the entered credentials are authenticated, a cookie is added for the logged in user and the browser redirects to the dashboard.

|  |  |  |
| --- | --- | --- |
| Object | Data Type | Reason |
| Cookie | HttpCookie | To keep a user logged in |
| Redirect | bool | To tell the JavaScript if redirection is required |
| Username | string | Stores username |
| Password | string | Stores password |

## Sign Up

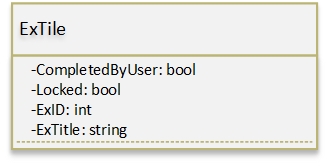
The SignUp page works similarly to the Login page in that the user put in data into textboxes which are transitted to the server via HTTP POST. Expect the SignUp page has more data entries, email, name and box for reentering the password. The box for reentering the password is to avoid the user mistyping their password during sign up and being locked out their account because of it. After the form post, the name, username and email are validated using regex strings.

|  |  |
| --- | --- |
| Use | Regex Pattern |
| Name | [a-zA-Z'-] |
| Username | ^[a-zA-Z\_0-9]+([-+.'][a-zA-Z\_0-9]+)\*@[a-zA-Z\_0-9]+([-.][a-zA-Z\_0-9]+)\*.[a-zA-Z\_0-9]+([-.][a-zA-Z\_0-9]+)\*$ |
| Email | [a-zA-Z0-9'-\_.] |

|  |  |  |
| --- | --- | --- |
| Object | Data Type | Reason |
| Cookie | HttpCookie | To keep a user logged in |
| Redirect | bool | To tell the JavaScript if redirection is required |
| Username | string | Stores username |
| Password | string | Stores password |
| Name | string | Stores name |
| Email | string | Stores email |

## Dashboard

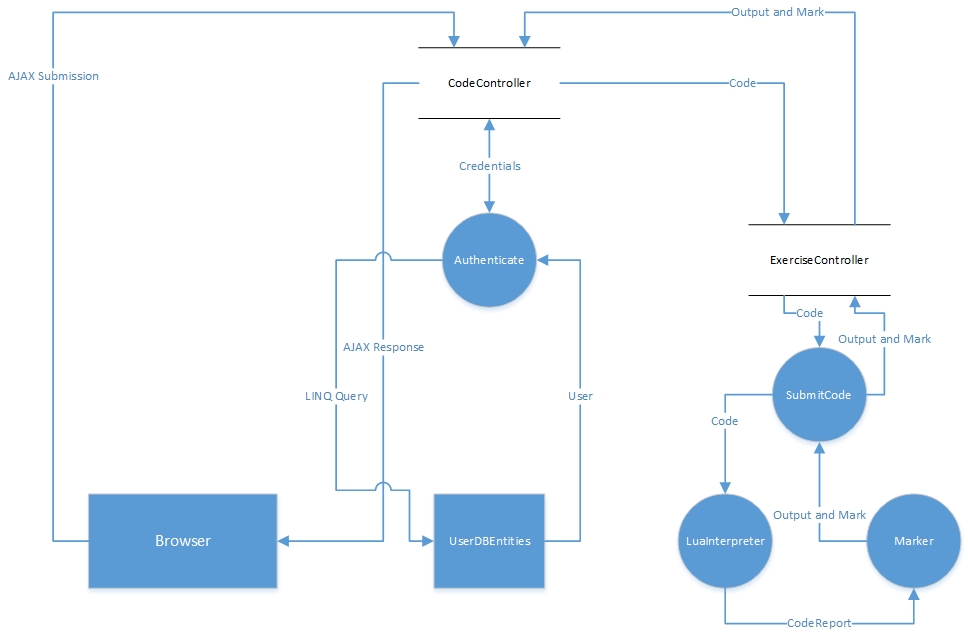
The dashboard is the homepage of the website and is also where the user picks what exercise they are going to do next. Each exercise is made into an ExTile object. An array of ExTiles is created and then listed on the dash. Below is a UML diagram for the ExTile class.



If the CompletedByUser field is true, the tile has a grey background on the dash. If the Locked field is true, the tile’s ‘Try Exercise’ button is disabled and cannot be clicked.

|  |  |  |
| --- | --- | --- |
| Object | Data Type | Reason |
| Cookie | HttpCookie | To keep a user logged in |
| Redirect | bool | To tell the JavaScript if redirection is required |
| Index | int | Index element of array |
| ExDB | ExDBEntities | To interact with exercise database |
| UserDB | UserDBEntities | To interact with user database |
| ExerciseTiles | ExTile[] | To store the ExTiles |

## Exercise



Above is a diagram that shows the flow of data and information between the user’s browser, the server’s modules and the databases.

### Interpreter

The interpreter is the largest and one of the most important parts of the system. It consists of two namespaces, Language.Lua and Language.Lua.Library. The libraries are in the Language.Lua.Library. There are three libraries:

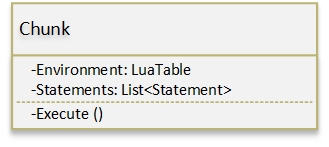
* BaseLib
* MathLib
* StringLib

BaseLib contains main functions used by the interpreter but also has functions like ‘print()’ and other type conversion methods used by the user.

MathLib contains functions and fields relating to math and numbers. It contains constant mathematical numbers such as π and e. It also has some functions that achieve tasks such as square root, power, and generate a random number. These are here to aid the user in their programming although not all are used in the exercises written.

StringLib contains functions for the manipulation of strings. Such functions are ‘len’ for getting the length of strings and ‘format’ for formatting and concatenating strings.

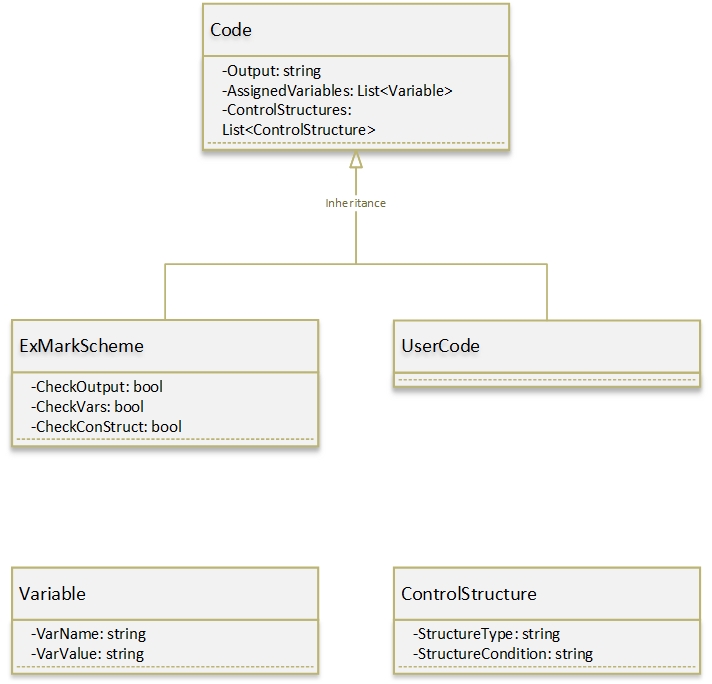
The Language.Lua namespace has the main internals of the interpreter in it. When the user’s code is passed to the LuaInterpreter, the interpreter creates the Global Environment. The Global Environment is the instance of the LuaTable class that contains the Lua libraries and modules. Once the Global Environment is created, the interpreter begins to parse the user’s code. The parsed code is broken down into tokens or ‘statements’. A statement is a representation of an operation in the Lua language e.g. an if statement, a while loop, etc. A list of statements is created; they are then executed in sequence. In the event of a syntax error, an exception is thrown before the statements are executed. During parsing, aspects of the user’s code are added to the CodeReport object. These aspects include the program output, variables and control structures.



### Marker

The Marker has two main classes involved, UserCode and ExMarkScheme. The UserCode is stored in an UserCode object. ExMarkScheme objects are stored in the database and created by the ExerciseParser when parsing an exercise file.

When marking the code, the mark scheme is retrieved from the database and used in the marking process. The marking process involves comparing the program output, variables and control structures.



## AddExercise

The AddExercise page is for admins to upload new exercises to the server. The view itself has a very plain and bland design. When you first visit the site, you are prompted by a message box asking you if you want to continue to an admin-only page. If the user declines, the user is redirected to another page. The page lists all the exercises in the database. It does this by using a foreach loop to progress through an array of ExRows. The ExRow class is what is printed in a list on the page.



When the file is uploaded through the HTML file upload control, it is sent via HTTP POST to the server. Currently, the user can only upload one exercise at a time. If I had of had more time, I would have liked to give the ability to be able to upload multiple files at a time and have them all processed together.

### ExerciseParser

Exercises are written in ‘.ex’ files. When the .ex file is uploaded, the server parses it and turns it into something the server understands and stores it in the database. The ExerciseParser class features a method called ParseExFile, this takes the filename of the uploaded exercise file as a parameter. The ParseExFile calls functions in the ExParsing class. The ExParsing class has functions that return different sections of the exercise file. The ExParsing class also contains an XMLParsing class, this has a function called ParseXML that returns an ExMarkScheme object. The ParseExFile method accumulates all the parsed data into an Exercise object. This object is then added to the ExDBEntities database and the database is saved asynchronously.

### Exercise File

An exercise file is a way of representing an Exercise object in text form. The exercise file format makes it easy for an admin to write an exercise and add it to the site quickly.

The exercise file consists of six main sections:

* ExID
* ExTitle
* ExDescription
* ExCodeTemplate
* ExAppendCode
* ExMarkScheme (XML representation)

Below is a template for an exercise file:

[ExID]

[ExTitle]

[ExDescription]

[ExCodeTemplate]

[ExAppendCode]

[ExMarkScheme]

<?xml version="1.0" encoding="utf-8"?>

<MarkScheme>

<Output></Output>

<Variables>

</Variables>

<ControlStructures>

</ControlStructures>

</MarkScheme>

The data required for each field is put on the line after the [Identifier].

Below is an example of an exercise file. This is the first exercise on the course, ‘Printing’:

[ExID]

1

[ExTitle]

Printing

[ExDescription]

Welcome to Lua. In this exercise, you are going to learn to print to the console using the print() function.

In Lua, we are able to display text to our users by using a command called print. This can be used like so:

print("Hello, User!")

In the code editor, try printing 'Hello world!' to the console.

[ExCodeTemplate]

[ExAppendCode]

[ExMarkScheme]

<?xml version="1.0" encoding="utf-8"?>

<MarkScheme>

<Output>Hello world!</Output>

</MarkScheme>

# Appraisal

My project was affected by time constraints in many places. If I had of had more time, I would have done the following things:

* Added more exercises that include tuition on functions, for loops and error handling.
* Improved the marker to allow for the new exercises I wanted to add.
* Made the website look better by using a nicer design.
* Added functionality in the AddExercise page to upload multiple exercise files at once and have them be parsed, consecutively and perhaps even concurrently.
* Move as much functionality as possible to the client side to reduce the server-side load.
* Improve security by implementing an SSL certificate so transmission can be through HTTPS.
* Add a caching system for the exercise page, currently, the loading of the page can be very slow due to the need to retrieve the exercise from the database.
* Fix incompatibility with Internet Explorer and Safari browsers.
* Add stricter permission functionality to website so only admins can upload exercises.

|  |  |
| --- | --- |
| Objective | Comment |
| *The system should be able to interpret the user’s code and mark it sufficiently.* | For the exercises written, this objective was met. The interpreter returns the program output and whether the code is what is expected depending on the exercise. |
| *The system should run the user’s code in a window on the browser.* | The user’s code is run in a textbox is run in a textbox on the browser that changes color depending on the correctness of the run code. |
| *The system should store the users’ details in a secure and professional fashion.* | The system stores the users’ data in MSSQL 2014 Server. The SQL Server is run on a Windows Server machine that is in a very secure network with Cisco firewalls and other threat preventing mechanisms. The password of the user is hashed twice using an SHA512 hash salted with username and email. Hashes are theoretically irreversible so a hacker could not find out the user’s password even if it got into the database. This objective was met. |
| *The system should parse exercises and store them correctly.* | An admin can upload an exercise and it is uploaded via HTTP POST and have it saved, parsed and stored in the database by the server. This objective was met. |
| *The system should be secure and be as invulnerable to attack as possible.* | In continuation of the points made on the 3rd objective analysis, the system encrypts the hashed password when it is stored in the cookie and transmitted via AJAX for the code interpretation, this adds an extra layer of security to the system. This objective was met. |
| *Communications should be as impervious to attack as possible.* | See the analysis of the 3rd and 5th objective. It should also be noted that all communication to the server is done through the Apollo VPN, the most secure VPN in the world. I think this objective was met. |
| *Interpreting and marking of code should be done quickly. Ideally, no greater than 55-60ms.* | As you can see from the picture above, the waiting time for the client before download of the response is 59.13ms. TTFB stands for Time To First Byte, meaning the aspects that make up the waiting time are the latency of server to client communication, marking, interpreting, authenticating user and packaging of the response object. So the interpreting is just a factor of the 59.16ms. In reality, the interpreting is less than 59.16ms so this objective has been met. |

# Appendix

## Test Evidence

|  |  |
| --- | --- |
| Test Number | Test Screenshot |
| 1.1 |  |
| 1.2 |  |
| 1.3 |  |
| 1.4 |  |
| 1.5 |  |
| 1.6 |  |
|  |  |
| 2.1 |  |
| 2.2 |  |
| 2.3 |  |
|  |  |
| 3.1 |  |
| 3.2 |  |
|  |  |
| 4.1 |  |
| 4.2 |  |

## Code

I am going to list the code with a heading of the folder name and then another heading above each code file.

Global.asax.cs

using Code\_College.App\_Start;

using System.Web.Http;

using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

namespace Code\_College

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

AreaRegistration.RegisterAllAreas();

GlobalConfiguration.Configure(WebApiConfig.Register);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

}

packages.config

<?xml version="1.0" encoding="utf-8"?>

<packages>

<package id="Antlr" version="3.5.0.2" targetFramework="net461" />

<package id="bootstrap" version="3.3.4" targetFramework="net461" />

<package id="EntityFramework" version="6.1.3" targetFramework="net452" />

<package id="jQuery" version="1.11.2" targetFramework="net461" />

<package id="Microsoft.AspNet.Mvc" version="5.2.3" targetFramework="net452" />

<package id="Microsoft.AspNet.Razor" version="3.2.3" targetFramework="net452" />

<package id="Microsoft.AspNet.Web.Optimization" version="1.1.3" targetFramework="net461" />

<package id="Microsoft.AspNet.WebApi" version="5.2.3" targetFramework="net46" />

<package id="Microsoft.AspNet.WebApi.Client" version="5.2.3" targetFramework="net46" />

<package id="Microsoft.AspNet.WebApi.Core" version="5.2.3" targetFramework="net46" />

<package id="Microsoft.AspNet.WebApi.WebHost" version="5.2.3" targetFramework="net46" />

<package id="Microsoft.AspNet.WebPages" version="3.2.3" targetFramework="net452" />

<package id="Microsoft.CodeDom.Providers.DotNetCompilerPlatform" version="1.0.1" targetFramework="net461" />

<package id="Microsoft.Net.Compilers" version="1.2.1" targetFramework="net46" developmentDependency="true" />

<package id="Microsoft.Owin" version="3.0.1" targetFramework="net461" />

<package id="Microsoft.Owin.Security" version="3.0.1" targetFramework="net461" />

<package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net452" />

<package id="Newtonsoft.Json" version="8.0.3" targetFramework="net46" />

<package id="Owin" version="1.0" targetFramework="net461" />

<package id="WebGrease" version="1.6.0" targetFramework="net461" />

</packages>

Web.config

<?xml version="1.0" encoding="utf-8"?>

<!--

For more information on how to configure your ASP.NET application, please visit

http://go.microsoft.com/fwlink/?LinkId=301880

-->

<configuration>

<configSections>

<!-- For more information on Entity Framework configuration, visit http://go.microsoft.com/fwlink/?LinkID=237468 -->

<section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection, EntityFramework, Version=6.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" requirePermission="false" />

</configSections>

<appSettings>

<add key="webpages:Version" value="3.0.0.0" />

<add key="webpages:Enabled" value="false" />

<add key="ClientValidationEnabled" value="true" />

<add key="UnobtrusiveJavaScriptEnabled" value="true" />

</appSettings>

<!--

For a description of web.config changes see http://go.microsoft.com/fwlink/?LinkId=235367.

The following attributes can be set on the <httpRuntime> tag.

<system.Web>

<httpRuntime targetFramework="4.6.1" />

</system.Web>

-->

<system.web>

<compilation debug="true" targetFramework="4.6" />

<httpRuntime />

<customErrors mode="Off" />

<pages controlRenderingCompatibilityVersion="4.0" />

</system.web>

<runtime>

<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">

<dependentAssembly>

<assemblyIdentity name="System.Web.Helpers" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.WebPages" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-3.0.0.0" newVersion="3.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="System.Web.Mvc" publicKeyToken="31bf3856ad364e35" />

<bindingRedirect oldVersion="1.0.0.0-5.2.3.0" newVersion="5.2.3.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="WebGrease" publicKeyToken="31bf3856ad364e35" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-1.6.5135.21930" newVersion="1.6.5135.21930" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Newtonsoft.Json" publicKeyToken="30ad4fe6b2a6aeed" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-8.0.0.0" newVersion="8.0.0.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin" publicKeyToken="31bf3856ad364e35" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.Owin.Security" publicKeyToken="31bf3856ad364e35" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-3.0.1.0" newVersion="3.0.1.0" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Antlr3.Runtime" publicKeyToken="eb42632606e9261f" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-3.5.0.2" newVersion="3.5.0.2" />

</dependentAssembly>

<dependentAssembly>

<assemblyIdentity name="Microsoft.AspNet.SignalR.Core" publicKeyToken="31bf3856ad364e35" culture="neutral" />

<bindingRedirect oldVersion="0.0.0.0-2.2.0.0" newVersion="2.2.0.0" />

</dependentAssembly>

</assemblyBinding>

</runtime>

<entityFramework>

<defaultConnectionFactory type="System.Data.Entity.Infrastructure.LocalDbConnectionFactory, EntityFramework">

<parameters>

<parameter value="MSSQLSERVER" />

</parameters>

</defaultConnectionFactory>

<providers>

<provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework.SqlServer" />

</providers>

</entityFramework>

<connectionStrings>

<add name="UserDBEntities" providerName="System.Data.SqlClient" connectionString="Data Source=RYAN-HP;Initial Catalog=UserDBEntities;Integrated Security=True;MultipleActiveResultSets=True" />

<add name="ExDBEntities" providerName="System.Data.SqlClient" connectionString="Data Source=RYAN-HP;Initial Catalog=ExDBEntities;Integrated Security=True;MultipleActiveResultSets=True" />

</connectionStrings>

<system.codedom>

<compilers>

<compiler language="c#;cs;csharp" extension=".cs" type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4" compilerOptions="/langversion:6 /nowarn:1659;1699;1701" />

<compiler language="vb;vbs;visualbasic;vbscript" extension=".vb" type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform, Version=1.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35" warningLevel="4" compilerOptions="/langversion:14 /nowarn:41008 /define:\_MYTYPE=\&quot;Web\&quot; /optionInfer+" />

</compilers>

</system.codedom>

<system.webServer>

<handlers>

<remove name="ExtensionlessUrlHandler-Integrated-4.0" />

<remove name="OPTIONSVerbHandler" />

<remove name="TRACEVerbHandler" />

<add name="ExtensionlessUrlHandler-Integrated-4.0" path="\*." verb="GET,HEAD,POST,DEBUG,PUT,DELETE" type="System.Web.Handlers.TransferRequestHandler" preCondition="integratedMode,runtimeVersionv4.0" />

</handlers>

</system.webServer>

</configuration>

### App\_Start

BundleConfig.cs

using System.Web.Optimization;

namespace Code\_College.App\_Start

{

// Set up bundling

public class BundleConfig

{

public static void RegisterBundles(BundleCollection Bundles)

{

// Enable optimizations that remove whitespace and size of files without compromising functionality

BundleTable.EnableOptimizations = true;

// Login and sign up bundles

ScriptBundle ScriptBundle = new ScriptBundle("~/bundles/js");

StyleBundle StyleBundle = new StyleBundle("~/bundles/css");

// Editor bundles for code editor in exercise view

ScriptBundle EditorScripts = new ScriptBundle("~/bundles/editorjs");

StyleBundle EditorStyles = new StyleBundle("~/bundles/editorcss");

ScriptBundle.IncludeDirectory("~/Scripts/", "\*.js");

StyleBundle.IncludeDirectory("~/Content/", "\*.css");

EditorScripts.IncludeDirectory("~/Scripts/Editor/", "\*.js");

EditorStyles.IncludeDirectory("~/Content/Editor/", "\*.css");

EditorStyles.IncludeDirectory("~/Content/Editor/Themes/", "\*.css");

Bundles.Add(ScriptBundle);

Bundles.Add(StyleBundle);

Bundles.Add(EditorScripts);

Bundles.Add(EditorStyles);

}

}

}

RouteConfig.cs

using System.Web.Mvc;

using System.Web.Routing;

namespace Code\_College

{

// Set up MVC routing

public class RouteConfig

{

public static void RegisterRoutes(RouteCollection routes)

{

routes.IgnoreRoute("{resource}.axd/{\*pathInfo}");

routes.MapRoute(

name: "Default",

url: "{controller}/{action}/{id}",

defaults: new { controller = "Dashboard", action = "Index", id = UrlParameter.Optional }

);

}

}

}

WebApiConfig.cs

using System.Web.Http;

namespace Code\_College.App\_Start

{

// Set up Web API routing

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

// Web API routes

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

}

### Controllers

AddExerciseController.cs

using Code\_College.Models;

using System;

using System.IO;

using System.Web;

using System.Web.Mvc;

namespace Code\_College.Controllers

{

public class AddExerciseController : Controller

{

public static ExRow[] ExerciseRows;

private static ExDBEntities ExDB = new ExDBEntities();

// GET: AddExercise

public ActionResult Index()

{

// Determines how many exercises are in the database and creates an array of that size

int NoOfExercises = 0;

foreach (Exercise Exercise in ExDB.Exercises)

NoOfExercises++;

ExerciseRows = new ExRow[NoOfExercises];

return View();

}

// Handles file submission

[HttpPost]

public ActionResult Index(HttpPostedFileBase File)

{

// Checks to see if the file is empty

if (File.ContentLength > 0)

{

// Saves file in an 'Exercises' directory on the server

string FileName = Path.GetFileName(File.FileName);

string FilePath = Path.Combine(Server.MapPath("~/Exercises"), FileName);

// Checks to see if file is an exercise file

if (File.FileName.EndsWith(".ex"))

{

File.SaveAs(FilePath);

ExerciseParser.ParseExFile(FilePath);

}

}

return RedirectToAction("Index");

}

public class ExRow

{

public int ExID { get; set; }

public string ExTitle { get; set; }

}

}

}

CodeController.cs

using Code\_College.Models;

using Language.Lua;

using System;

using System.Linq;

using System.Net;

using System.Web.Http;

namespace Code\_College.Controllers

{

public class CodeController : ApiController

{

private static ExDBEntities ExDB = new ExDBEntities();

private static UserDBEntities UserDB = new UserDBEntities();

// POST api/<controller>

[HttpPost]

public string PostSubmission(Submission Submission)

{

// Double checks the user that uploaded the code is a valid user

if (!Authenticate(Submission.username, Submission.password))

throw new HttpResponseException(HttpStatusCode.Unauthorized);

Exercise CurrentExercise = ExDB.Exercises.Where(x => x.ExID == Submission.id).FirstOrDefault();

// Checks to see if the exercise is a valid exercise and throws a HTTP 404 if not

if (CurrentExercise == null)

throw new HttpResponseException(HttpStatusCode.NotFound);

// Returns output of the program

return SubmitCode(Submission.code, CurrentExercise, Submission.username);

}

// Authenticates the user

private bool Authenticate(string Username, string Password)

{

User user = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (user.PasswordHash == Account.HashCredentials(user.Email, Account.Cryptography.Decrypt(Password, Username)))

return true;

else

return false;

}

// Handles interpretation of code and returns the output of the code

public static string SubmitCode(string Code, Exercise CurrentExercise, string Username)

{

bool Correct;

string ConsoleOutput = "Sorry, that was incorrect. Please, read the task and try again.";

Code += CurrentExercise.ExAppendCode ?? "";

Marker.Marker.MarkScheme = CurrentExercise.ExMarkScheme;

try

{

LuaInterpreter.RunCode(Code);

Correct = Marker.Marker.FullMark();

}

catch (Exception ex)

{

Correct = false;

ConsoleOutput = ex.Message;

}

if (Correct)

{

Account.LevelUp(Username, CurrentExercise);

ConsoleOutput = LuaInterpreter.CodeReport.Output;

return ConsoleOutput;

}

else

return ConsoleOutput;

}

}

public class Submission

{

public string code { get; set; }

public int id { get; set; }

public string password { get; set; }

public string username { get; set; }

}

}

DashboardController.cs

using Code\_College.Models;

using System.Web.Mvc;

namespace Code\_College.Controllers

{

public class DashboardController : Controller

{

public static ExTile[] ExerciseTiles;

private static ExDBEntities ExDB = new ExDBEntities();

// GET: Dashboard

public ActionResult Index()

{

// Determines how many exercises are in the database and creates an array of that size

int NoOfExercises = 0;

foreach (Exercise Exercise in ExDB.Exercises)

NoOfExercises++;

ExerciseTiles = new ExTile[NoOfExercises];

return View();

}

public class ExTile

{

public bool CompletedByUser { get; set; }

public bool Locked { get; set; }

public int ExID { get; set; }

public string ExTitle { get; set; }

}

}

}

ExerciseController.cs

using Code\_College.Models;

using System.Linq;

using System.Web.Mvc;

namespace Code\_College.Controllers

{

public class ExerciseController : Controller

{

private static ExDBEntities ExDB = new ExDBEntities();

// GET: Exercise

public ActionResult Index(int id = 1)

{

Exercise CurrentExercise = ExDB.Exercises.Where(x => x.ExID == id).FirstOrDefault();

// Loads exercise data into the ViewBag for the view to display

ViewBag.ExID = CurrentExercise.ExID;

ViewBag.Title = CurrentExercise.ExTitle + " - Code College";

ViewBag.ExerciseTitle = CurrentExercise.ExTitle;

ViewBag.Desc = CurrentExercise.ExDescription;

ViewBag.ExerciseID = CurrentExercise.ExID;

ViewBag.CodeTemplate = CurrentExercise.ExCodeTemplate ?? "";

ViewBag.AppendCode = CurrentExercise.ExAppendCode ?? "";

return View();

}

}

}

LoginController.cs

using System.Web.Mvc;

namespace Code\_College.Controllers

{

public class LoginController : Controller

{

// GET: Login

public ActionResult Index()

{

return View();

}

}

}

SignUpController.cs

using System.Web.Mvc;

namespace Code\_College.Controllers

{

public class LoginController : Controller

{

// GET: Login

public ActionResult Index()

{

return View();

}

}

}

### Interpreter

Chunk.cs

using Marker;

using System;

namespace Language.Lua

{

public partial class Assignment : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

LuaValue[] values = ExprList.ConvertAll(expr => expr.Evaluate(environment)).ToArray();

LuaValue[] neatValues = LuaMultiValue.UnWrapLuaValues(values);

for (int i = 0; i < Math.Min(VarList.Count, neatValues.Length); i++)

{

Var var = VarList[i];

if (var.Accesses.Count == 0)

{

VarName varName = var.Base as VarName;

if (varName != null)

{

SetKeyValue(environment, new LuaString(varName.Name), values[i]);

LuaInterpreter.CodeReport.AssignedVariables.Add(new UserCode.Variable { VarName = varName.Name, VarValue = values[i].Value.ToString() });

continue;

}

}

else

{

LuaValue baseValue = var.Base.Evaluate(environment);

for (int j = 0; j < var.Accesses.Count - 1; j++)

{

Access access = var.Accesses[j];

baseValue = access.Evaluate(baseValue, environment);

}

Access lastAccess = var.Accesses[var.Accesses.Count - 1];

NameAccess nameAccess = lastAccess as NameAccess;

if (nameAccess != null)

{

SetKeyValue(baseValue, new LuaString(nameAccess.Name), values[i]);

continue;

}

KeyAccess keyAccess = lastAccess as KeyAccess;

if (lastAccess != null)

{

SetKeyValue(baseValue, keyAccess.Key.Evaluate(environment), values[i]);

}

}

}

isBreak = false;

return null;

}

private static void SetKeyValue(LuaValue baseValue, LuaValue key, LuaValue value)

{

LuaValue newIndex = LuaNil.Nil;

LuaTable table = baseValue as LuaTable;

if (table != null)

{

if (table.ContainsKey(key))

{

table.SetKeyValue(key, value);

return;

}

else

{

if (table.MetaTable != null)

{

newIndex = table.MetaTable.GetValue("\_\_newindex");

}

if (newIndex == LuaNil.Nil)

{

table.SetKeyValue(key, value);

return;

}

}

}

else

{

LuaUserdata userdata = baseValue as LuaUserdata;

if (userdata != null)

{

if (userdata.MetaTable != null)

{

newIndex = userdata.MetaTable.GetValue("\_\_newindex");

}

if (newIndex == LuaNil.Nil)

{

throw new Exception("Interpreter: Assign field of userdata without \_\_newindex defined.");

}

}

}

LuaFunction func = newIndex as LuaFunction;

if (func != null)

{

func.Invoke(new LuaValue[] { baseValue, key, value });

}

else

{

SetKeyValue(newIndex, key, value);

}

}

}

public partial class BreakStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

throw new NotImplementedException();

}

}

public partial class Chunk

{

public LuaTable Environment;

public LuaValue Execute()

{

bool isBreak;

return Execute(out isBreak);

}

public LuaValue Execute(LuaTable environment, out bool isBreak)

{

Environment = new LuaTable(environment);

return Execute(out isBreak);

}

public LuaValue Execute(out bool isBreak)

{

foreach (Statement statement in Statements)

{

ReturnStmt returnStmt = statement as ReturnStmt;

if (returnStmt != null)

{

isBreak = false;

return LuaMultiValue.WrapLuaValues(returnStmt.ExprList.ConvertAll(expr => expr.Evaluate(Environment)).ToArray());

}

else if (statement is BreakStmt)

{

isBreak = true;

return null;

}

else

{

var returnValue = statement.Execute(Environment, out isBreak);

if (returnValue != null || isBreak == true)

{

return returnValue;

}

}

}

isBreak = false;

return null;

}

}

public partial class DoStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

return Body.Execute(environment, out isBreak);

}

}

public partial class ExprStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

Expr.Evaluate(environment);

isBreak = false;

return null;

}

}

public partial class ForInStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

LuaValue[] values = ExprList.ConvertAll(expr => expr.Evaluate(environment)).ToArray();

LuaValue[] neatValues = LuaMultiValue.UnWrapLuaValues(values);

LuaFunction func = neatValues[0] as LuaFunction;

LuaValue state = neatValues[1];

LuaValue loopVar = neatValues[2];

var table = new LuaTable(environment);

Body.Environment = table;

while (true)

{

LuaValue result = func.Invoke(new LuaValue[] { state, loopVar });

LuaMultiValue multiValue = result as LuaMultiValue;

if (multiValue != null)

{

neatValues = LuaMultiValue.UnWrapLuaValues(multiValue.Values);

loopVar = neatValues[0];

for (int i = 0; i < Math.Min(NameList.Count, neatValues.Length); i++)

{

table.SetNameValue(NameList[i], neatValues[i]);

}

}

else

{

loopVar = result;

table.SetNameValue(NameList[0], result);

}

if (loopVar == LuaNil.Nil)

{

break;

}

var returnValue = Body.Execute(out isBreak);

if (returnValue != null || isBreak == true)

{

isBreak = false;

return returnValue;

}

}

isBreak = false;

return null;

}

}

public partial class ForStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

LuaNumber start = Start.Evaluate(environment) as LuaNumber;

LuaNumber end = End.Evaluate(environment) as LuaNumber;

double step = 1;

if (Step != null)

{

step = (Step.Evaluate(environment) as LuaNumber).Number;

}

var table = new LuaTable(environment);

table.SetNameValue(VarName, start);

Body.Environment = table;

while (step > 0 && start.Number <= end.Number ||

step <= 0 && start.Number >= end.Number)

{

var returnValue = Body.Execute(out isBreak);

if (returnValue != null || isBreak == true)

{

isBreak = false;

return returnValue;

}

start.Number += step;

}

isBreak = false;

return null;

}

}

public partial class Function : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

LuaTable table = environment;

if (Name.MethodName == null)

{

for (int i = 0; i < Name.FullName.Count - 1; i++)

{

LuaValue obj = environment.GetValue(Name.FullName[i]);

table = obj as LuaTable;

if (table == null)

{

throw new Exception("Interpreter: Not a table: " + Name.FullName[i]);

}

}

table.SetNameValue(

Name.FullName[Name.FullName.Count - 1],

Body.Evaluate(environment));

}

else

{

for (int i = 0; i < Name.FullName.Count; i++)

{

LuaValue obj = environment.GetValue(Name.FullName[i]);

table = obj as LuaTable;

if (table == null)

{

throw new Exception("Interpreter: Not a table " + Name.FullName[i]);

}

}

Body.ParamList.NameList.Insert(0, "self");

table.SetNameValue(

Name.MethodName,

Body.Evaluate(environment));

}

isBreak = false;

return null;

}

}

public partial class IfStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

LuaValue condition = Condition.Evaluate(environment);

if (condition.GetBooleanValue() == true)

{

return ThenBlock.Execute(environment, out isBreak);

}

else

{

foreach (ElseifBlock elseifBlock in ElseifBlocks)

{

condition = elseifBlock.Condition.Evaluate(environment);

if (condition.GetBooleanValue() == true)

{

return elseifBlock.ThenBlock.Execute(environment, out isBreak);

}

}

if (ElseBlock != null)

{

return ElseBlock.Execute(environment, out isBreak);

}

}

isBreak = false;

return null;

}

}

public partial class LocalFunc : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

environment.SetNameValue(Name, Body.Evaluate(environment));

isBreak = false;

return null;

}

}

public partial class LocalVar : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

LuaValue[] values = ExprList.ConvertAll(expr => expr.Evaluate(environment)).ToArray();

LuaValue[] neatValues = LuaMultiValue.UnWrapLuaValues(values);

for (int i = 0; i < Math.Min(NameList.Count, neatValues.Length); i++)

{

environment.RawSetValue(NameList[i], neatValues[i]);

}

if (neatValues.Length < NameList.Count)

{

for (int i = neatValues.Length; i < NameList.Count - neatValues.Length; i++)

{

environment.RawSetValue(NameList[i], LuaNil.Nil);

}

}

isBreak = false;

return null;

}

}

public partial class RepeatStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

while (true)

{

var returnValue = Body.Execute(environment, out isBreak);

if (returnValue != null || isBreak == true)

{

isBreak = false;

return returnValue;

}

LuaValue condition = Condition.Evaluate(environment);

if (condition.GetBooleanValue() == true)

{

break;

}

}

return null;

}

}

public partial class ReturnStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

throw new NotImplementedException();

}

}

public abstract partial class Statement

{

public abstract LuaValue Execute(LuaTable environment, out bool isBreak);

}

public partial class WhileStmt : Statement

{

public override LuaValue Execute(LuaTable environment, out bool isBreak)

{

while (true)

{

LuaValue condition = Condition.Evaluate(environment);

if (condition.GetBooleanValue() == false)

{

break;

}

var returnValue = Body.Execute(environment, out isBreak);

if (returnValue != null || isBreak == true)

{

isBreak = false;

return returnValue;

}

}

isBreak = false;

return null;

}

}

}

Expr.cs

using System;

using System.Collections.Generic;

using System.Globalization;

namespace Language.Lua

{

public enum Associativity

{

NonAssociative,

LeftAssociative,

RightAssociative

}

public abstract partial class Access

{

public abstract LuaValue Evaluate(LuaValue baseValue, LuaTable environment);

}

public abstract partial class BaseExpr : Term

{

}

public partial class BoolLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return LuaBoolean.From(bool.Parse(Text));

}

}

public abstract partial class Expr

{

public abstract LuaValue Evaluate(LuaTable environment);

public abstract Term Simplify();

}

public partial class FunctionBody

{

public LuaValue Evaluate(LuaTable environment)

{

return new LuaFunction(

new LuaFunc(delegate (LuaValue[] args)

{

var table = new LuaTable(environment);

List<string> names = ParamList.NameList;

if (names.Count > 0)

{

int argCount = Math.Min(names.Count, args.Length);

for (int i = 0; i < argCount; i++)

{

table.SetNameValue(names[i], args[i]);

}

if (ParamList.HasVarArg)

{

if (argCount < args.Length)

{

LuaValue[] remainedArgs = new LuaValue[args.Length - argCount];

for (int i = 0; i < remainedArgs.Length; i++)

{

remainedArgs[i] = args[argCount + i];

}

table.SetNameValue("...", new LuaMultiValue(remainedArgs));

}

}

}

else if (ParamList.IsVarArg != null)

{

table.SetNameValue("...", new LuaMultiValue(args));

}

Chunk.Environment = table;

return Chunk.Execute();

})

);

}

}

public partial class FunctionCall : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaFunction function = baseValue as LuaFunction;

if (function != null)

{

if (function.Function.Method.DeclaringType.FullName == "Language.Lua.Library.BaseLib" &&

(function.Function.Method.Name == "loadstring" || function.Function.Method.Name == "dofile"))

{

if (Args.String != null)

{

return function.Function.Invoke(new LuaValue[] { Args.String.Evaluate(environment), environment });

}

else

{

return function.Function.Invoke(new LuaValue[] { Args.ArgList[0].Evaluate(environment), environment });

}

}

if (Args.Table != null)

{

return function.Function.Invoke(new LuaValue[] { Args.Table.Evaluate(environment) });

}

else if (Args.String != null)

{

return function.Function.Invoke(new LuaValue[] { Args.String.Evaluate(environment) });

}

else

{

List<LuaValue> args = Args.ArgList.ConvertAll(arg => arg.Evaluate(environment));

return function.Function.Invoke(LuaMultiValue.UnWrapLuaValues(args.ToArray()));

}

}

else

{

throw new Exception("Interpreter: Invoke function call on non function value.");

}

}

}

public partial class FunctionValue : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return Body.Evaluate(environment);

}

}

public partial class GroupExpr : BaseExpr

{

public override LuaValue Evaluate(LuaTable environment)

{

return Expr.Evaluate(environment);

}

public override Term Simplify()

{

return Expr.Simplify();

}

}

public partial class KeyAccess : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaValue key = Key.Evaluate(environment);

return LuaValue.GetKeyValue(baseValue, key);

}

}

public partial class MethodCall : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaValue value = LuaValue.GetKeyValue(baseValue, new LuaString(Method));

LuaFunction function = value as LuaFunction;

if (function != null)

{

if (Args.Table != null)

{

return function.Function.Invoke(new LuaValue[] { baseValue, Args.Table.Evaluate(environment) });

}

else if (Args.String != null)

{

return function.Function.Invoke(new LuaValue[] { baseValue, Args.String.Evaluate(environment) });

}

else

{

List<LuaValue> args = Args.ArgList.ConvertAll(arg => arg.Evaluate(environment));

args.Insert(0, baseValue);

return function.Function.Invoke(args.ToArray());

}

}

else

{

throw new Exception("Interpreter: Invoke method call on non function value.");

}

}

}

public partial class NameAccess : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaValue key = new LuaString(Name);

return LuaValue.GetKeyValue(baseValue, key);

}

}

public partial class NilLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return LuaNil.Nil;

}

}

public partial class NumberLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

double number;

if (string.IsNullOrEmpty(HexicalText))

{

number = double.Parse(Text, NumberStyles.AllowDecimalPoint | NumberStyles.AllowExponent | NumberStyles.AllowLeadingSign);

}

else

{

number = int.Parse(HexicalText, NumberStyles.HexNumber);

}

return new LuaNumber(number);

}

}

/// <summary>

/// Represent Unary or Binary Operation, for Unary Operation the LeftOperand is not used.

/// </summary>

public partial class Operation : Term

{

public Term LeftOperand;

public string Operator;

public Term RightOperand;

public Operation(string oper)

{

Operator = oper;

}

public Operation(string oper, Term left, Term right)

{

Operator = oper;

LeftOperand = left == null ? null : left.Simplify();

RightOperand = right == null ? null : right.Simplify();

}

public override LuaValue Evaluate(LuaTable environment)

{

if (LeftOperand == null)

{

return PrefixUnaryOperation(Operator, RightOperand, environment);

}

else if (RightOperand == null)

{

return LeftOperand.Evaluate(environment);

}

else

{

return InfixBinaryOperation(LeftOperand, Operator, RightOperand, environment);

}

}

private static int? Compare(LuaValue leftValue, LuaValue rightValue)

{

LuaNumber left = leftValue as LuaNumber;

LuaNumber right = rightValue as LuaNumber;

if (left != null && right != null)

{

return left.Number.CompareTo(right.Number);

}

LuaString leftString = leftValue as LuaString;

LuaString rightString = rightValue as LuaString;

if (leftString != null && rightString != null)

{

return StringComparer.Ordinal.Compare(leftString.Text, rightString.Text);

}

return null;

}

private static LuaFunction GetMetaFunction(string name, LuaValue leftValue, LuaValue rightValue)

{

LuaTable left = leftValue as LuaTable;

if (left != null)

{

LuaFunction func = left.GetValue(name) as LuaFunction;

if (func != null)

{

return func;

}

}

LuaTable right = rightValue as LuaTable;

if (right != null)

{

return right.GetValue(name) as LuaFunction;

}

return null;

}

private LuaValue InfixBinaryOperation(Term LeftOperand, string Operator, Term RightOperand, LuaTable environment)

{

LuaValue leftValue = LeftOperand.Evaluate(environment);

LuaValue rightValue = RightOperand.Evaluate(environment);

switch (Operator)

{

case "+":

var left = leftValue as LuaNumber;

var right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number + right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_add", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "-":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number - right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_sub", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "\*":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number \* right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_mul", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "/":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number / right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_div", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "%":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number % right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_mod", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "^":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(Math.Pow(left.Number, right.Number));

}

else

{

LuaFunction func = GetMetaFunction("\_\_pow", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "==":

return LuaBoolean.From(leftValue.Equals(rightValue));

case "~=":

return LuaBoolean.From(leftValue.Equals(rightValue) == false);

case "<":

int? compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare < 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_lt", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case ">":

compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare > 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_gt", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "<=":

compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare <= 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_le", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case ">=":

compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare >= 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_ge", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "..":

if ((leftValue is LuaString || leftValue is LuaNumber) &&

(rightValue is LuaString || rightValue is LuaNumber))

{

return new LuaString(string.Concat(leftValue, rightValue));

}

else

{

LuaFunction func = GetMetaFunction("\_\_concat", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "and":

bool leftBool = leftValue.GetBooleanValue();

bool rightBool = rightValue.GetBooleanValue();

if (leftBool == false)

{

return leftValue;

}

else

{

return rightValue;

}

case "or":

leftBool = leftValue.GetBooleanValue();

rightBool = rightValue.GetBooleanValue();

if (leftBool == true)

{

return leftValue;

}

else

{

return rightValue;

}

}

return null;

}

private LuaValue PrefixUnaryOperation(string Operator, Term RightOperand, LuaTable environment)

{

LuaValue rightValue = RightOperand.Evaluate(environment);

switch (Operator)

{

case "-":

var number = rightValue as LuaNumber;

if (number != null)

{

return new LuaNumber(-number.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_unm", rightValue, null);

if (func != null)

{

return func.Invoke(new LuaValue[] { rightValue });

}

}

break;

case "#":

var table = rightValue as LuaTable;

if (table != null)

{

return new LuaNumber(table.Length);

}

var str = rightValue as LuaString;

if (str != null)

{

return new LuaNumber(str.Text.Length);

}

break;

case "not":

var rightBool = rightValue as LuaBoolean;

if (rightBool != null)

{

return LuaBoolean.From(!rightBool.BoolValue);

}

break;

}

return LuaNil.Nil;

}

}

public partial class OperatorExpr : Expr

{

public LinkedList<object> Terms = new LinkedList<object>();

public void Add(string oper)

{

Terms.AddLast(oper);

}

public void Add(Term term)

{

Terms.AddLast(term);

}

public Term BuildExpressionTree()

{

var node = Terms.First;

Term term = node.Value as Term;

if (Terms.Count == 1)

{

return term;

}

else

{

if (term != null)

{

return BuildExpressionTree(node.Value as Term, node.Next);

}

string oper = node.Value as string;

if (oper != null)

{

return BuildExpressionTree(null, node);

}

return null;

}

}

public override LuaValue Evaluate(LuaTable environment)

{

Term term = BuildExpressionTree();

return term.Evaluate(environment);

}

public override Term Simplify()

{

return BuildExpressionTree().Simplify();

}

// Operator-precedence parsing algorithm

private static Term BuildExpressionTree(Term leftTerm, LinkedListNode<object> node)

{

string oper = node.Value as string;

var rightNode = node.Next;

Term rightTerm = rightNode.Value as Term;

if (rightNode.Next == null) // last node

{

return new Operation(oper, leftTerm, rightTerm);

}

else

{

string nextOper = rightNode.Next.Value as string;

if (OperTable.IsPrior(oper, nextOper))

{

return BuildExpressionTree(new Operation(oper, leftTerm, rightTerm), rightNode.Next);

}

else

{

return new Operation(oper, leftTerm, BuildExpressionTree(rightTerm, rightNode.Next));

}

}

}

}

public class OperTable

{

private static Associativity[] associativity;

private static Dictionary<string, int> precedence = new Dictionary<string, int>();

static OperTable()

{

List<string[]> operators = new List<string[]>();

operators.Add(new string[] { "or" });

operators.Add(new string[] { "and" });

operators.Add(new string[] { "==", "~=" });

operators.Add(new string[] { ">", ">=", "<", "<=" });

operators.Add(new string[] { ".." });

operators.Add(new string[] { "+", "-" });

operators.Add(new string[] { "\*", "/", "%" });

operators.Add(new string[] { "#", "not" });

operators.Add(new string[] { "^" });

for (int index = 0; index < operators.Count; index++)

{

foreach (string oper in operators[index])

{

precedence.Add(oper, index);

}

}

associativity = new Associativity[operators.Count];

associativity[0] = Associativity.LeftAssociative;

associativity[1] = Associativity.LeftAssociative;

associativity[2] = Associativity.NonAssociative;

associativity[3] = Associativity.LeftAssociative;

associativity[4] = Associativity.LeftAssociative;

associativity[5] = Associativity.LeftAssociative;

associativity[6] = Associativity.LeftAssociative;

associativity[7] = Associativity.NonAssociative;

associativity[8] = Associativity.RightAssociative;

}

/// <summary>

/// Whether the input text is an operator or not

/// </summary>

/// <param name="oper"></param>

/// <returns></returns>

public static bool Contains(string oper)

{

return precedence.ContainsKey(oper);

}

/// <summary>

/// whether operLeft has higher precedence than operRight

/// </summary>

/// <param name="operLeft"></param>

/// <param name="operRight"></param>

/// <returns></returns>

public static bool IsPrior(string operLeft, string operRight)

{

if (operLeft == null) return false;

if (operRight == null) return true;

int priLeft = precedence[operLeft];

int priRight = precedence[operRight];

if (priLeft > priRight)

{

return true;

}

else if (priLeft < priRight)

{

return false;

}

else

{

switch (associativity[priLeft])

{

case Associativity.LeftAssociative:

return true;

case Associativity.RightAssociative:

return false;

default:

return true;

}

}

}

}

public partial class PrimaryExpr : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

LuaValue baseValue = Base.Evaluate(environment);

foreach (Access access in Accesses)

{

baseValue = access.Evaluate(baseValue, environment);

}

return baseValue;

}

public override Term Simplify()

{

if (Accesses.Count == 0)

{

return Base.Simplify();

}

else

{

return this;

}

}

}

public partial class StringLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return new LuaString(Text);

}

}

public partial class TableConstructor : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

LuaTable table = new LuaTable();

foreach (Field field in FieldList)

{

NameValue nameValue = field as NameValue;

if (nameValue != null)

{

table.SetNameValue(nameValue.Name, nameValue.Value.Evaluate(environment));

continue;

}

KeyValue keyValue = field as KeyValue;

if (keyValue != null)

{

table.SetKeyValue(

keyValue.Key.Evaluate(environment),

keyValue.Value.Evaluate(environment));

continue;

}

ItemValue itemValue = field as ItemValue;

if (itemValue != null)

{

table.AddValue(itemValue.Value.Evaluate(environment));

continue;

}

}

return table;

}

}

public partial class Term : Expr

{

public override LuaValue Evaluate(LuaTable environment)

{

throw new NotImplementedException();

}

public override Term Simplify()

{

return this;

}

}

public partial class VariableArg : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return environment.GetValue(Name);

}

}

public partial class VarName : BaseExpr

{

public override LuaValue Evaluate(LuaTable environment)

{

return environment.GetValue(Name);

}

public override Term Simplify()

{

return this;

}

}

}

LuaInterpreter.cs

using System;

using System.Collections.Generic;

using System.Globalization;

namespace Language.Lua

{

public enum Associativity

{

NonAssociative,

LeftAssociative,

RightAssociative

}

public abstract partial class Access

{

public abstract LuaValue Evaluate(LuaValue baseValue, LuaTable environment);

}

public abstract partial class BaseExpr : Term

{

}

public partial class BoolLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return LuaBoolean.From(bool.Parse(Text));

}

}

public abstract partial class Expr

{

public abstract LuaValue Evaluate(LuaTable environment);

public abstract Term Simplify();

}

public partial class FunctionBody

{

public LuaValue Evaluate(LuaTable environment)

{

return new LuaFunction(

new LuaFunc(delegate (LuaValue[] args)

{

var table = new LuaTable(environment);

List<string> names = ParamList.NameList;

if (names.Count > 0)

{

int argCount = Math.Min(names.Count, args.Length);

for (int i = 0; i < argCount; i++)

{

table.SetNameValue(names[i], args[i]);

}

if (ParamList.HasVarArg)

{

if (argCount < args.Length)

{

LuaValue[] remainedArgs = new LuaValue[args.Length - argCount];

for (int i = 0; i < remainedArgs.Length; i++)

{

remainedArgs[i] = args[argCount + i];

}

table.SetNameValue("...", new LuaMultiValue(remainedArgs));

}

}

}

else if (ParamList.IsVarArg != null)

{

table.SetNameValue("...", new LuaMultiValue(args));

}

Chunk.Environment = table;

return Chunk.Execute();

})

);

}

}

public partial class FunctionCall : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaFunction function = baseValue as LuaFunction;

if (function != null)

{

if (function.Function.Method.DeclaringType.FullName == "Language.Lua.Library.BaseLib" &&

(function.Function.Method.Name == "loadstring" || function.Function.Method.Name == "dofile"))

{

if (Args.String != null)

{

return function.Function.Invoke(new LuaValue[] { Args.String.Evaluate(environment), environment });

}

else

{

return function.Function.Invoke(new LuaValue[] { Args.ArgList[0].Evaluate(environment), environment });

}

}

if (Args.Table != null)

{

return function.Function.Invoke(new LuaValue[] { Args.Table.Evaluate(environment) });

}

else if (Args.String != null)

{

return function.Function.Invoke(new LuaValue[] { Args.String.Evaluate(environment) });

}

else

{

List<LuaValue> args = Args.ArgList.ConvertAll(arg => arg.Evaluate(environment));

return function.Function.Invoke(LuaMultiValue.UnWrapLuaValues(args.ToArray()));

}

}

else

{

throw new Exception("Interpreter: Invoke function call on non function value.");

}

}

}

public partial class FunctionValue : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return Body.Evaluate(environment);

}

}

public partial class GroupExpr : BaseExpr

{

public override LuaValue Evaluate(LuaTable environment)

{

return Expr.Evaluate(environment);

}

public override Term Simplify()

{

return Expr.Simplify();

}

}

public partial class KeyAccess : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaValue key = Key.Evaluate(environment);

return LuaValue.GetKeyValue(baseValue, key);

}

}

public partial class MethodCall : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaValue value = LuaValue.GetKeyValue(baseValue, new LuaString(Method));

LuaFunction function = value as LuaFunction;

if (function != null)

{

if (Args.Table != null)

{

return function.Function.Invoke(new LuaValue[] { baseValue, Args.Table.Evaluate(environment) });

}

else if (Args.String != null)

{

return function.Function.Invoke(new LuaValue[] { baseValue, Args.String.Evaluate(environment) });

}

else

{

List<LuaValue> args = Args.ArgList.ConvertAll(arg => arg.Evaluate(environment));

args.Insert(0, baseValue);

return function.Function.Invoke(args.ToArray());

}

}

else

{

throw new Exception("Interpreter: Invoke method call on non function value.");

}

}

}

public partial class NameAccess : Access

{

public override LuaValue Evaluate(LuaValue baseValue, LuaTable environment)

{

LuaValue key = new LuaString(Name);

return LuaValue.GetKeyValue(baseValue, key);

}

}

public partial class NilLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return LuaNil.Nil;

}

}

public partial class NumberLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

double number;

if (string.IsNullOrEmpty(HexicalText))

{

number = double.Parse(Text, NumberStyles.AllowDecimalPoint | NumberStyles.AllowExponent | NumberStyles.AllowLeadingSign);

}

else

{

number = int.Parse(HexicalText, NumberStyles.HexNumber);

}

return new LuaNumber(number);

}

}

/// <summary>

/// Represent Unary or Binary Operation, for Unary Operation the LeftOperand is not used.

/// </summary>

public partial class Operation : Term

{

public Term LeftOperand;

public string Operator;

public Term RightOperand;

public Operation(string oper)

{

Operator = oper;

}

public Operation(string oper, Term left, Term right)

{

Operator = oper;

LeftOperand = left == null ? null : left.Simplify();

RightOperand = right == null ? null : right.Simplify();

}

public override LuaValue Evaluate(LuaTable environment)

{

if (LeftOperand == null)

{

return PrefixUnaryOperation(Operator, RightOperand, environment);

}

else if (RightOperand == null)

{

return LeftOperand.Evaluate(environment);

}

else

{

return InfixBinaryOperation(LeftOperand, Operator, RightOperand, environment);

}

}

private static int? Compare(LuaValue leftValue, LuaValue rightValue)

{

LuaNumber left = leftValue as LuaNumber;

LuaNumber right = rightValue as LuaNumber;

if (left != null && right != null)

{

return left.Number.CompareTo(right.Number);

}

LuaString leftString = leftValue as LuaString;

LuaString rightString = rightValue as LuaString;

if (leftString != null && rightString != null)

{

return StringComparer.Ordinal.Compare(leftString.Text, rightString.Text);

}

return null;

}

private static LuaFunction GetMetaFunction(string name, LuaValue leftValue, LuaValue rightValue)

{

LuaTable left = leftValue as LuaTable;

if (left != null)

{

LuaFunction func = left.GetValue(name) as LuaFunction;

if (func != null)

{

return func;

}

}

LuaTable right = rightValue as LuaTable;

if (right != null)

{

return right.GetValue(name) as LuaFunction;

}

return null;

}

private LuaValue InfixBinaryOperation(Term LeftOperand, string Operator, Term RightOperand, LuaTable environment)

{

LuaValue leftValue = LeftOperand.Evaluate(environment);

LuaValue rightValue = RightOperand.Evaluate(environment);

switch (Operator)

{

case "+":

var left = leftValue as LuaNumber;

var right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number + right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_add", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "-":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number - right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_sub", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "\*":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number \* right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_mul", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "/":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number / right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_div", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "%":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(left.Number % right.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_mod", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "^":

left = leftValue as LuaNumber;

right = rightValue as LuaNumber;

if (left != null && right != null)

{

return new LuaNumber(Math.Pow(left.Number, right.Number));

}

else

{

LuaFunction func = GetMetaFunction("\_\_pow", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "==":

return LuaBoolean.From(leftValue.Equals(rightValue));

case "~=":

return LuaBoolean.From(leftValue.Equals(rightValue) == false);

case "<":

int? compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare < 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_lt", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case ">":

compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare > 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_gt", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "<=":

compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare <= 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_le", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case ">=":

compare = Compare(leftValue, rightValue);

if (compare != null)

{

return LuaBoolean.From(compare >= 0);

}

else

{

LuaFunction func = GetMetaFunction("\_\_ge", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "..":

if ((leftValue is LuaString || leftValue is LuaNumber) &&

(rightValue is LuaString || rightValue is LuaNumber))

{

return new LuaString(string.Concat(leftValue, rightValue));

}

else

{

LuaFunction func = GetMetaFunction("\_\_concat", leftValue, rightValue);

if (func != null)

{

return func.Invoke(new LuaValue[] { leftValue, rightValue });

}

}

break;

case "and":

bool leftBool = leftValue.GetBooleanValue();

bool rightBool = rightValue.GetBooleanValue();

if (leftBool == false)

{

return leftValue;

}

else

{

return rightValue;

}

case "or":

leftBool = leftValue.GetBooleanValue();

rightBool = rightValue.GetBooleanValue();

if (leftBool == true)

{

return leftValue;

}

else

{

return rightValue;

}

}

return null;

}

private LuaValue PrefixUnaryOperation(string Operator, Term RightOperand, LuaTable environment)

{

LuaValue rightValue = RightOperand.Evaluate(environment);

switch (Operator)

{

case "-":

var number = rightValue as LuaNumber;

if (number != null)

{

return new LuaNumber(-number.Number);

}

else

{

LuaFunction func = GetMetaFunction("\_\_unm", rightValue, null);

if (func != null)

{

return func.Invoke(new LuaValue[] { rightValue });

}

}

break;

case "#":

var table = rightValue as LuaTable;

if (table != null)

{

return new LuaNumber(table.Length);

}

var str = rightValue as LuaString;

if (str != null)

{

return new LuaNumber(str.Text.Length);

}

break;

case "not":

var rightBool = rightValue as LuaBoolean;

if (rightBool != null)

{

return LuaBoolean.From(!rightBool.BoolValue);

}

break;

}

return LuaNil.Nil;

}

}

public partial class OperatorExpr : Expr

{

public LinkedList<object> Terms = new LinkedList<object>();

public void Add(string oper)

{

Terms.AddLast(oper);

}

public void Add(Term term)

{

Terms.AddLast(term);

}

public Term BuildExpressionTree()

{

var node = Terms.First;

Term term = node.Value as Term;

if (Terms.Count == 1)

{

return term;

}

else

{

if (term != null)

{

return BuildExpressionTree(node.Value as Term, node.Next);

}

string oper = node.Value as string;

if (oper != null)

{

return BuildExpressionTree(null, node);

}

return null;

}

}

public override LuaValue Evaluate(LuaTable environment)

{

Term term = BuildExpressionTree();

return term.Evaluate(environment);

}

public override Term Simplify()

{

return BuildExpressionTree().Simplify();

}

// Operator-precedence parsing algorithm

private static Term BuildExpressionTree(Term leftTerm, LinkedListNode<object> node)

{

string oper = node.Value as string;

var rightNode = node.Next;

Term rightTerm = rightNode.Value as Term;

if (rightNode.Next == null) // last node

{

return new Operation(oper, leftTerm, rightTerm);

}

else

{

string nextOper = rightNode.Next.Value as string;

if (OperTable.IsPrior(oper, nextOper))

{

return BuildExpressionTree(new Operation(oper, leftTerm, rightTerm), rightNode.Next);

}

else

{

return new Operation(oper, leftTerm, BuildExpressionTree(rightTerm, rightNode.Next));

}

}

}

}

public class OperTable

{

private static Associativity[] associativity;

private static Dictionary<string, int> precedence = new Dictionary<string, int>();

static OperTable()

{

List<string[]> operators = new List<string[]>();

operators.Add(new string[] { "or" });

operators.Add(new string[] { "and" });

operators.Add(new string[] { "==", "~=" });

operators.Add(new string[] { ">", ">=", "<", "<=" });

operators.Add(new string[] { ".." });

operators.Add(new string[] { "+", "-" });

operators.Add(new string[] { "\*", "/", "%" });

operators.Add(new string[] { "#", "not" });

operators.Add(new string[] { "^" });

for (int index = 0; index < operators.Count; index++)

{

foreach (string oper in operators[index])

{

precedence.Add(oper, index);

}

}

associativity = new Associativity[operators.Count];

associativity[0] = Associativity.LeftAssociative;

associativity[1] = Associativity.LeftAssociative;

associativity[2] = Associativity.NonAssociative;

associativity[3] = Associativity.LeftAssociative;

associativity[4] = Associativity.LeftAssociative;

associativity[5] = Associativity.LeftAssociative;

associativity[6] = Associativity.LeftAssociative;

associativity[7] = Associativity.NonAssociative;

associativity[8] = Associativity.RightAssociative;

}

/// <summary>

/// Whether the input text is an operator or not

/// </summary>

/// <param name="oper"></param>

/// <returns></returns>

public static bool Contains(string oper)

{

return precedence.ContainsKey(oper);

}

/// <summary>

/// whether operLeft has higher precedence than operRight

/// </summary>

/// <param name="operLeft"></param>

/// <param name="operRight"></param>

/// <returns></returns>

public static bool IsPrior(string operLeft, string operRight)

{

if (operLeft == null) return false;

if (operRight == null) return true;

int priLeft = precedence[operLeft];

int priRight = precedence[operRight];

if (priLeft > priRight)

{

return true;

}

else if (priLeft < priRight)

{

return false;

}

else

{

switch (associativity[priLeft])

{

case Associativity.LeftAssociative:

return true;

case Associativity.RightAssociative:

return false;

default:

return true;

}

}

}

}

public partial class PrimaryExpr : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

LuaValue baseValue = Base.Evaluate(environment);

foreach (Access access in Accesses)

{

baseValue = access.Evaluate(baseValue, environment);

}

return baseValue;

}

public override Term Simplify()

{

if (Accesses.Count == 0)

{

return Base.Simplify();

}

else

{

return this;

}

}

}

public partial class StringLiteral : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return new LuaString(Text);

}

}

public partial class TableConstructor : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

LuaTable table = new LuaTable();

foreach (Field field in FieldList)

{

NameValue nameValue = field as NameValue;

if (nameValue != null)

{

table.SetNameValue(nameValue.Name, nameValue.Value.Evaluate(environment));

continue;

}

KeyValue keyValue = field as KeyValue;

if (keyValue != null)

{

table.SetKeyValue(

keyValue.Key.Evaluate(environment),

keyValue.Value.Evaluate(environment));

continue;

}

ItemValue itemValue = field as ItemValue;

if (itemValue != null)

{

table.AddValue(itemValue.Value.Evaluate(environment));

continue;

}

}

return table;

}

}

public partial class Term : Expr

{

public override LuaValue Evaluate(LuaTable environment)

{

throw new NotImplementedException();

}

public override Term Simplify()

{

return this;

}

}

public partial class VariableArg : Term

{

public override LuaValue Evaluate(LuaTable environment)

{

return environment.GetValue(Name);

}

}

public partial class VarName : BaseExpr

{

public override LuaValue Evaluate(LuaTable environment)

{

return environment.GetValue(Name);

}

public override Term Simplify()

{

return this;

}

}

}

#### LuaLibs

BaseLib.cs

using System;

namespace Language.Lua.Library

{

internal class BaseLib

{

// Lua assert function

public static LuaValue assert(LuaValue[] values)

{

bool condition = values[0].GetBooleanValue();

LuaString message = values.Length > 1 ? values[1] as LuaString : null;

if (message != null)

{

throw new LuaError("Interpreter: " + message.Text);

}

else

{

throw new LuaError("Interpreter: assertion failed!");

}

}

// Lua error function

public static LuaValue error(LuaValue[] values)

{

LuaString message = values[0] as LuaString;

if (message != null)

{

throw new LuaError("Interpreter: " + message.Text);

}

else

{

throw new LuaError("Interpreter: error raised!");

}

}

public static LuaValue getmetatable(LuaValue[] values)

{

LuaTable table = values[0] as LuaTable;

return table.MetaTable;

}

// Lua function for printing to the console

public static LuaValue print(LuaValue[] values)

{

LuaInterpreter.CodeReport.Output += string.Join<LuaValue>(" ", values);

return null;

}

// Registers functions in library

public static void RegisterFunctions(LuaTable module)

{

module.Register("print", print);

module.Register("type", type);

module.Register("getmetatable", getmetatable);

module.Register("setmetatable", setmetatable);

module.Register("tostring", tostring);

module.Register("tonumber", tonumber);

module.Register("assert", assert);

module.Register("error", error);

}

public static LuaValue setmetatable(LuaValue[] values)

{

LuaTable table = values[0] as LuaTable;

LuaTable metatable = values[1] as LuaTable;

table.MetaTable = metatable;

return null;

}

// Lua function for converting string to a number

public static LuaValue tonumber(LuaValue[] values)

{

LuaString text = values[0] as LuaString;

if (text != null)

{

return new LuaNumber(double.Parse(text.Text));

}

LuaString number = values[0] as LuaString;

if (number != null)

{

return number;

}

return LuaNil.Nil;

}

// Lua function for converting number to a string

public static LuaValue tostring(LuaValue[] values)

{

return new LuaString(values[0].ToString());

}

// Lua function that returns the data type of an object

public static LuaValue type(LuaValue[] values)

{

if (values.Length > 0)

{

return new LuaString(values[0].GetTypeCode());

}

else

{

throw new Exception("Interpreter: bad argument #1 to 'type' (value expected)");

}

}

}

}

MathLib.cs

using System;

namespace Language.Lua.Library

{

public static class MathLib

{

private static Random randomGenerator = new Random();

// Lua function for finding power of two numbers

public static LuaValue pow(LuaValue[] values)

{

var numbers = CheckArgs2(values);

return new LuaNumber(Math.Pow(numbers.Item1, numbers.Item2));

}

// Lua function for creating random number within limits

public static LuaValue random(LuaValue[] values)

{

if (values.Length == 0)

{

return new LuaNumber(randomGenerator.NextDouble());

}

else if (values.Length == 1)

{

LuaNumber number1 = values[0] as LuaNumber;

return new LuaNumber(randomGenerator.Next((int)number1.Number));

}

else

{

var numbers = CheckArgs2(values);

return new LuaNumber(randomGenerator.Next((int)numbers.Item1, (int)numbers.Item2));

}

}

// Lua function for creating random seed

public static LuaValue randomseed(LuaValue[] values)

{

LuaNumber number = CheckArgs(values);

randomGenerator = new Random((int)number.Number);

return number;

}

// Registers functions in library

public static void RegisterFunctions(LuaTable module)

{

module.SetNameValue("huge", new LuaNumber(double.MaxValue));

module.SetNameValue("pi", new LuaNumber(Math.PI));

module.SetNameValue("e", new LuaNumber(Math.E));

module.Register("pow", pow);

module.Register("random", random);

module.Register("randomseed", randomseed);

module.Register("sqrt", sqrt);

}

// Registers library as module in the environment

public static void RegisterModule(LuaTable environment)

{

LuaTable module = new LuaTable();

RegisterFunctions(module);

environment.SetNameValue("math", module);

}

// Lua function for finding the square root of a number

public static LuaValue sqrt(LuaValue[] values)

{

LuaNumber number = CheckArgs(values);

return new LuaNumber(Math.Sqrt(number.Number));

}

private static LuaNumber CheckArgs(LuaValue[] values)

{

if (values.Length >= 1)

{

LuaNumber number = values[0] as LuaNumber;

if (number != null)

{

return number;

}

else

{

throw new LuaError("Interpreter: bad argument #1 to 'abs' (number expected, got {0})", values[0].GetTypeCode());

}

}

else

{

throw new LuaError("Interpreter: bad argument #1 to 'abs' (number expected, got no value)");

}

}

private static Tuple<double, double> CheckArgs2(LuaValue[] values)

{

if (values.Length >= 2)

{

LuaNumber number1 = values[0] as LuaNumber;

if (number1 == null)

{

throw new LuaError("Interpreter: bad argument #1 to 'abs' (number expected, got {0})", values[0].GetTypeCode());

}

LuaNumber number2 = values[1] as LuaNumber;

if (number2 == null)

{

throw new LuaError("Interpreter: bad argument #2 to 'abs' (number expected, got {0})", values[1].GetTypeCode());

}

return Tuple.Create(number1.Number, number2.Number);

}

else

{

throw new LuaError("Interpreter: bad argument #1 to 'abs' (number expected, got no value)");

}

}

}

}

StringLib.cs

namespace Language.Lua.Library

{

public static class StringLib

{

// Lua function for formatting strings

public static LuaValue format(LuaValue[] values)

{

LuaString format = values[0] as LuaString;

object[] args = new object[values.Length - 1];

for (int i = 0; i < args.Length; i++)

{

args[i] = values[i + 1].Value;

}

return new LuaString(string.Format(format.Text, args));

}

// Lua function for finding length of string

public static LuaValue len(LuaValue[] values)

{

LuaString str = values[0] as LuaString;

return new LuaNumber(str.Text.Length);

}

// Registers functions in library

public static void RegisterFunctions(LuaTable module)

{

module.Register("format", format);

module.Register("len", len);

}

// Registers library as module in the environment

public static void RegisterModule(LuaTable environment)

{

LuaTable module = new LuaTable();

RegisterFunctions(module);

environment.SetNameValue("string", module);

}

}

}

#### LuaValue

LuaBoolean.cs

namespace Language.Lua

{

public class LuaBoolean : LuaValue

{

public static readonly LuaBoolean False = new LuaBoolean { BoolValue = false };

public static readonly LuaBoolean True = new LuaBoolean { BoolValue = true };

private LuaBoolean()

{

}

public bool BoolValue { get; set; }

public override object Value

{

get { return BoolValue; }

}

public static LuaBoolean From(bool value)

{

if (value == true)

return True;

else

return False;

}

public override bool GetBooleanValue()

{

return BoolValue;

}

public override string GetTypeCode()

{

return "boolean";

}

public override string ToString()

{

return BoolValue.ToString().ToLower();

}

}

}

LuaError.cs

using System;

namespace Language.Lua

{

public class LuaError : Exception

{

public LuaError(string message)

: base(message)

{

}

public LuaError(string message, Exception innerException)

: base(message, innerException)

{

}

public LuaError(string messageformat, params object[] args)

: base(string.Format(messageformat, args))

{

}

}

}

LuaFunction.cs

namespace Language.Lua

{

public delegate LuaValue LuaFunc(LuaValue[] args);

public class LuaFunction : LuaValue

{

public LuaFunction(LuaFunc function)

{

Function = function;

}

public LuaFunc Function { get; set; }

public override object Value

{

get { return Function; }

}

public override string GetTypeCode()

{

return "function";

}

public LuaValue Invoke(LuaValue[] args)

{

return Function.Invoke(args);

}

}

}

LuaMultiValue.cs

using System;

using System.Collections.Generic;

namespace Language.Lua

{

public class LuaMultiValue : LuaValue

{

public LuaMultiValue(LuaValue[] values)

{

Values = values;

}

public override object Value

{

get { return Values; }

}

public LuaValue[] Values { get; set; }

public static LuaValue[] UnWrapLuaValues(LuaValue[] values)

{

if (values == null || values.Length == 0 || ContainsMultiValue(values) == false)

return values;

if (values.Length == 1 && values[0] is LuaMultiValue)

return (values[0] as LuaMultiValue).Values;

List<LuaValue> neatValues = new List<LuaValue>(values.Length);

for (int i = 0; i < values.Length - 1; i++)

{

LuaValue value = values[i];

LuaMultiValue multiValue = value as LuaMultiValue;

if (multiValue != null)

neatValues.Add(multiValue.Values[0]);

else

neatValues.Add(value);

}

LuaValue lastValue = values[values.Length - 1];

LuaMultiValue lastMultiValue = lastValue as LuaMultiValue;

if (lastMultiValue != null)

neatValues.AddRange(lastMultiValue.Values);

else

neatValues.Add(lastValue);

return neatValues.ToArray();

}

public static LuaValue WrapLuaValues(LuaValue[] values)

{

if (values == null || values.Length == 0)

return LuaNil.Nil;

else if (values.Length == 1)

return values[0];

else

return new LuaMultiValue(UnWrapLuaValues(values));

}

public override string GetTypeCode()

{

throw new InvalidOperationException();

}

private static bool ContainsMultiValue(LuaValue[] values)

{

foreach (LuaValue value in values)

{

if (value is LuaMultiValue)

return true;

}

return false;

}

}

}

LuaNil.cs

namespace Language.Lua

{

public class LuaNil : LuaValue

{

public static readonly LuaNil Nil = new LuaNil();

private LuaNil()

{

}

public override object Value

{

get { return null; }

}

public override bool GetBooleanValue()

{

return false;

}

public override string GetTypeCode()

{

return "nil";

}

public override string ToString()

{

return "nil";

}

}

}

LuaNumber.cs

namespace Language.Lua

{

public class LuaNumber : LuaValue

{

public LuaNumber(double number)

{

Number = number;

}

public double Number { get; set; }

public override object Value

{

get { return Number; }

}

public override string GetTypeCode()

{

return "number";

}

public override string ToString()

{

return Number.ToString();

}

}

}

LuaString.cs

namespace Language.Lua

{

public class LuaString : LuaValue

{

public static readonly LuaString Empty = new LuaString(string.Empty);

public LuaString(string text)

{

Text = text;

}

public string Text { get; set; }

public override object Value

{

get { return Text; }

}

public override string GetTypeCode()

{

return "string";

}

public override string ToString()

{

return Text;

}

}

}

LuaTable.cs

using System;

using System.Collections.Generic;

namespace Language.Lua

{

public class LuaTable : LuaValue

{

private Dictionary<LuaValue, LuaValue> dict;

private List<LuaValue> list;

public LuaTable()

{

}

public LuaTable(LuaTable parent)

{

MetaTable = new LuaTable();

MetaTable.SetNameValue("\_\_index", parent);

MetaTable.SetNameValue("\_\_newindex", parent);

}

public int Count

{

get

{

if (dict == null)

return 0;

else

return dict.Count;

}

}

public IEnumerable<LuaValue> Keys

{

get

{

if (Length > 0)

{

for (int index = 1; index <= list.Count; index++)

yield return new LuaNumber(index);

}

if (Count > 0)

{

foreach (LuaValue key in dict.Keys)

yield return key;

}

}

}

public IEnumerable<KeyValuePair<LuaValue, LuaValue>> KeyValuePairs

{

get { return dict; }

}

public int Length

{

get

{

if (list == null)

{

return 0;

}

else

{

return list.Count;

}

}

}

public IEnumerable<LuaValue> ListValues

{

get { return list; }

}

public LuaTable MetaTable { get; set; }

public override object Value

{

get { return this; }

}

public void AddValue(LuaValue value)

{

if (list == null)

{

list = new List<LuaValue>();

}

list.Add(value);

}

public bool ContainsKey(LuaValue key)

{

if (dict != null)

{

if (dict.ContainsKey(key))

return true;

}

if (list != null)

{

LuaNumber index = key as LuaNumber;

if (index != null && index.Number == (int)index.Number)

return index.Number >= 1 && index.Number <= list.Count;

}

return false;

}

public LuaValue GetKey(string key)

{

if (dict == null) return LuaNil.Nil;

foreach (LuaValue value in dict.Keys)

{

LuaString str = value as LuaString;

if (str != null && string.Equals(str.Text, key, StringComparison.Ordinal))

return value;

}

return LuaNil.Nil;

}

public override string GetTypeCode()

{

return "table";

}

public LuaValue GetValue(int index)

{

if (index > 0 && index <= Length)

return list[index - 1];

return LuaNil.Nil;

}

public LuaValue GetValue(string name)

{

LuaValue key = GetKey(name);

if (key == LuaNil.Nil)

{

if (MetaTable != null)

return GetValueFromMetaTable(name);

return LuaNil.Nil;

}

else

return dict[key];

}

public LuaValue GetValue(LuaValue key)

{

if (key == LuaNil.Nil)

return LuaNil.Nil;

else

{

LuaNumber number = key as LuaNumber;

if (number != null && number.Number == (int)number.Number)

{

int index = (int)number.Number;

if (index > 0 && index <= Length)

return list[index - 1];

}

if (dict != null && dict.ContainsKey(key))

return dict[key];

else if (MetaTable != null)

return GetValueFromMetaTable(key);

return LuaNil.Nil;

}

}

public void InsertValue(int index, LuaValue value)

{

if (index > 0 && index <= Length + 1)

list.Insert(index - 1, value);

else

throw new ArgumentOutOfRangeException("index");

}

public LuaValue RawGetValue(LuaValue key)

{

if (dict != null && dict.ContainsKey(key))

return dict[key];

return LuaNil.Nil;

}

public void RawSetValue(string name, LuaValue value)

{

LuaValue key = GetKey(name);

if (key == LuaNil.Nil)

key = new LuaString(name);

if (dict == null)

dict = new Dictionary<LuaValue, LuaValue>();

dict[key] = value;

}

public LuaFunction Register(string name, LuaFunc function)

{

LuaFunction luaFunc = new LuaFunction(function);

SetNameValue(name, luaFunc);

return luaFunc;

}

public bool Remove(LuaValue item)

{

return list.Remove(item);

}

public void RemoveAt(int index)

{

list.RemoveAt(index - 1);

}

public void SetKeyValue(LuaValue key, LuaValue value)

{

LuaNumber number = key as LuaNumber;

if (number != null && number.Number == (int)number.Number)

{

int index = (int)number.Number;

if (index == Length + 1)

{

AddValue(value);

return;

}

if (index > 0 && index <= Length)

{

list[index - 1] = value;

return;

}

}

if (value == LuaNil.Nil)

{

RemoveKey(key);

return;

}

if (dict == null)

dict = new Dictionary<LuaValue, LuaValue>();

dict[key] = value;

}

public void SetNameValue(string name, LuaValue value)

{

if (value == LuaNil.Nil)

RemoveKey(name);

else

RawSetValue(name, value);

}

public void Sort()

{

list.Sort((a, b) =>

{

LuaNumber n = a as LuaNumber;

LuaNumber m = b as LuaNumber;

if (n != null && m != null)

return n.Number.CompareTo(m.Number);

LuaString s = a as LuaString;

LuaString t = b as LuaString;

if (s != null && t != null)

return s.Text.CompareTo(t.Text);

return 0;

});

}

public void Sort(LuaFunction compare)

{

list.Sort((a, b) =>

{

LuaValue result = compare.Invoke(new LuaValue[] { a, b });

LuaBoolean boolValue = result as LuaBoolean;

if (boolValue != null && boolValue.BoolValue == true)

return 1;

else

return -1;

});

}

public override string ToString()

{

if (MetaTable != null)

{

LuaFunction function = MetaTable.GetValue("\_\_tostring") as LuaFunction;

if (function != null)

return function.Invoke(new LuaValue[] { this }).ToString();

}

return "Table " + GetHashCode();

}

private LuaValue GetValueFromMetaTable(string name)

{

LuaValue indexer = MetaTable.GetValue("\_\_index");

LuaTable table = indexer as LuaTable;

if (table != null)

return table.GetValue(name);

LuaFunction function = indexer as LuaFunction;

if (function != null)

return function.Function.Invoke(new LuaValue[] { new LuaString(name) });

return LuaNil.Nil;

}

private LuaValue GetValueFromMetaTable(LuaValue key)

{

LuaValue indexer = MetaTable.GetValue("\_\_index");

LuaTable table = indexer as LuaTable;

if (table != null)

return table.GetValue(key);

LuaFunction function = indexer as LuaFunction;

if (function != null)

return function.Function.Invoke(new LuaValue[] { key });

return LuaNil.Nil;

}

private void RemoveKey(string name)

{

LuaValue key = GetKey(name);

if (key != LuaNil.Nil)

dict.Remove(key);

}

private void RemoveKey(LuaValue key)

{

if (key != LuaNil.Nil && dict != null && dict.ContainsKey(key))

dict.Remove(key);

}

}

}

LuaUserData.cs

namespace Language.Lua

{

public class LuaUserdata : LuaValue

{

private object Object;

public LuaUserdata(object obj)

{

Object = obj;

}

public LuaUserdata(object obj, LuaTable metatable)

{

Object = obj;

MetaTable = metatable;

}

public LuaTable MetaTable { get; set; }

public override object Value

{

get { return Object; }

}

public override string GetTypeCode()

{

return "userdata";

}

public override string ToString()

{

return "userdata";

}

}

}

LuaValue.cs

using System;

namespace Language.Lua

{

public abstract class LuaValue : IEquatable<LuaValue>

{

public abstract object Value { get; }

public static LuaValue GetKeyValue(LuaValue baseValue, LuaValue key)

{

LuaTable table = baseValue as LuaTable;

if (table != null)

return table.GetValue(key);

else

{

LuaUserdata userdata = baseValue as LuaUserdata;

if (userdata != null)

{

if (userdata.MetaTable != null)

{

LuaValue index = userdata.MetaTable.GetValue("\_\_index");

if (index != null)

{

LuaFunction func = index as LuaFunction;

if (func != null)

return func.Invoke(new LuaValue[] { baseValue, key });

else

return GetKeyValue(index, key);

}

}

}

throw new Exception(string.Format("Interpreter: Access field '{0}' from not a table.", key.Value));

}

}

public bool Equals(LuaValue other)

{

if (other == null)

return false;

if (this is LuaNil)

return other is LuaNil;

if (this is LuaTable && other is LuaTable)

return ReferenceEquals(this, other);

return Value.Equals(other.Value);

}

public virtual bool GetBooleanValue()

{

return true;

}

public override int GetHashCode()

{

if (this is LuaNumber || this is LuaString)

return Value.GetHashCode();

return base.GetHashCode();

}

public abstract string GetTypeCode();

}

}

#### Parser

Syntax.cs

using System;

namespace Language.Lua

{

public abstract class LuaValue : IEquatable<LuaValue>

{

public abstract object Value { get; }

public static LuaValue GetKeyValue(LuaValue baseValue, LuaValue key)

{

LuaTable table = baseValue as LuaTable;

if (table != null)

return table.GetValue(key);

else

{

LuaUserdata userdata = baseValue as LuaUserdata;

if (userdata != null)

{

if (userdata.MetaTable != null)

{

LuaValue index = userdata.MetaTable.GetValue("\_\_index");

if (index != null)

{

LuaFunction func = index as LuaFunction;

if (func != null)

return func.Invoke(new LuaValue[] { baseValue, key });

else

return GetKeyValue(index, key);

}

}

}

throw new Exception(string.Format("Interpreter: Access field '{0}' from not a table.", key.Value));

}

}

public bool Equals(LuaValue other)

{

if (other == null)

return false;

if (this is LuaNil)

return other is LuaNil;

if (this is LuaTable && other is LuaTable)

return ReferenceEquals(this, other);

return Value.Equals(other.Value);

}

public virtual bool GetBooleanValue()

return true;

public override int GetHashCode()

{

if (this is LuaNumber || this is LuaString)

return Value.GetHashCode();

return base.GetHashCode();

}

public abstract string GetTypeCode();

}

}

Parser.cs

using System;

namespace Language.Lua

{

public abstract class LuaValue : IEquatable<LuaValue>

{

public abstract object Value { get; }

public static LuaValue GetKeyValue(LuaValue baseValue, LuaValue key)

{

LuaTable table = baseValue as LuaTable;

if (table != null)

return table.GetValue(key);

else

{

LuaUserdata userdata = baseValue as LuaUserdata;

if (userdata != null)

{

if (userdata.MetaTable != null)

{

LuaValue index = userdata.MetaTable.GetValue("\_\_index");

if (index != null)

{

LuaFunction func = index as LuaFunction;

if (func != null)

return func.Invoke(new LuaValue[] { baseValue, key });

else

return GetKeyValue(index, key);

}

}

}

throw new Exception(string.Format("Interpreter: Access field '{0}' from not a table.", key.Value));

}

}

public bool Equals(LuaValue other)

{

if (other == null)

return false;

if (this is LuaNil)

return other is LuaNil;

if (this is LuaTable && other is LuaTable)

return ReferenceEquals(this, other);

return Value.Equals(other.Value);

}

public virtual bool GetBooleanValue()

return true;

public override int GetHashCode()

{

if (this is LuaNumber || this is LuaString)

return Value.GetHashCode();

return base.GetHashCode();

}

public abstract string GetTypeCode();

}

}

ParserInput.cs

namespace Language.Lua

{

public interface ParserInput<T>

{

int Length { get; }

string FormErrorMessage(int position, string message);

T GetInputSymbol(int pos);

T[] GetSubSection(int position, int length);

bool HasInput(int pos);

}

}

TextInput.cs

using System.Collections.Generic;

namespace Language.Lua

{

public class TextInput : ParserInput<char>

{

private string InputText;

private List<int> LineBreaks;

public TextInput(string text)

{

InputText = text;

LineBreaks = new List<int>();

LineBreaks.Add(0);

for (int index = 0; index < InputText.Length; index++)

{

if (InputText[index] == '\n')

LineBreaks.Add(index + 1);

}

LineBreaks.Add(InputText.Length);

}

#region ParserInput<char> Members

public int Length

{

get { return InputText.Length; }

}

public string FormErrorMessage(int position, string message)

{

int line;

int col;

GetLineColumnNumber(position, out line, out col);

string ch = HasInput(position) ? "'" + GetInputSymbol(position) + "'" : null;

return string.Format("Line {0}, Col {1} {2}: {3}", line, col, ch, message);

}

public char GetInputSymbol(int pos)

{

return InputText[pos];

}

public char[] GetSubSection(int position, int length)

{

return InputText.Substring(position, length).ToCharArray();

}

public bool HasInput(int pos)

{

return pos < InputText.Length;

}

#endregion ParserInput<char> Members

public void GetLineColumnNumber(int pos, out int line, out int col)

{

col = 1;

for (line = 1; line < LineBreaks.Count; line++)

{

if (LineBreaks[line] > pos)

{

for (int p = LineBreaks[line - 1]; p < pos; p++)

{

if (InputText[p] == '\t')

col += 4;

else

col++;

}

break;

}

}

}

public string GetSubString(int start, int length)

{

return InputText.Substring(start, length);

}

}

}

TextParserCommon.cs

using System;

using System.Collections.Generic;

using System.Text;

namespace Language.Lua

{

public partial class Parser

{

public List<Tuple<int, string>> Errors = new List<Tuple<int, string>>();

private Stack<int> ErrorStack = new Stack<int>();

private ParserInput<char> Input;

/// <summary>

/// Memories parsing results, key is (PositionStart, Noterminal), value is (SyntacticElement, success, PostionAfter).

/// </summary>

private Dictionary<Tuple<int, string>, Tuple<object, bool, int>> ParsingResults = new Dictionary<Tuple<int, string>, Tuple<object, bool, int>>();

private int position;

public Parser()

{

}

public int Position

{

get { return position; }

set { position = value; }

}

public string GetErrorMessages()

{

StringBuilder text = new StringBuilder();

foreach (Tuple<int, string> msg in Errors)

{

text.Append(Input.FormErrorMessage(msg.Item1, msg.Item2));

text.AppendLine();

}

return text.ToString();

}

public void SetInput(ParserInput<char> input)

{

Input = input;

position = 0;

ParsingResults.Clear();

}

private void ClearError(int count)

{

Errors.RemoveRange(count, Errors.Count - count);

}

private int Error(string message)

{

Errors.Add(new Tuple<int, string>(position, message));

return Errors.Count;

}

private char MatchTerminal(char terminal, out bool success)

{

success = false;

if (Input.HasInput(position))

{

char symbol = Input.GetInputSymbol(position);

if (terminal == symbol)

{

position++;

success = true;

}

return symbol;

}

return default(char);

}

private char MatchTerminalRange(char start, char end, out bool success)

{

success = false;

if (Input.HasInput(position))

{

char symbol = Input.GetInputSymbol(position);

if (start <= symbol && symbol <= end)

{

position++;

success = true;

}

return symbol;

}

return default(char);

}

private char MatchTerminalSet(string terminalSet, bool isComplement, out bool success)

{

success = false;

if (Input.HasInput(position))

{

char symbol = Input.GetInputSymbol(position);

bool match = isComplement ? terminalSet.IndexOf(symbol) == -1 : terminalSet.IndexOf(symbol) > -1;

if (match)

{

position++;

success = true;

}

return symbol;

}

return default(char);

}

private string MatchTerminalString(string terminalString, out bool success)

{

int currrent\_position = position;

foreach (char terminal in terminalString)

{

MatchTerminal(terminal, out success);

if (!success)

{

position = currrent\_position;

return null;

}

}

success = true;

return terminalString;

}

private bool TerminalMatch(char terminal)

{

if (Input.HasInput(position))

{

char symbol = Input.GetInputSymbol(position);

return terminal == symbol;

}

return false;

}

private bool TerminalMatch(char terminal, int pos)

{

if (Input.HasInput(pos))

{

char symbol = Input.GetInputSymbol(pos);

return terminal == symbol;

}

return false;

}

}

}

### Marker

Marker.cs

using Language.Lua;

using System.Collections.Generic;

namespace Marker

{

public static class Marker

{

public static ExMarkScheme MarkScheme { get; set; }

public static bool FullMark()

{

List<bool> Marks = new List<bool>();

if (MarkScheme.CheckOutput)

Marks.Add(MarkOutput());

else if (MarkScheme.CheckVars)

Marks.Add(MarkVars());

else if (MarkScheme.CheckConStruct)

Marks.Add(MarkControlStructs());

else

return true;

foreach (bool Mark in Marks)

{

if (!Mark)

return false;

}

return true;

}

public static bool MarkControlStructs()

{

if (MarkScheme.ControlStructures.Contains(new ExMarkScheme.ControlStructure { StructureType = null, StructureCondition = null }))

return true;

bool Contains, DNM = false;

List<bool> Marks = new List<bool>();

foreach (UserCode.ControlStructure ConStruct in LuaInterpreter.CodeReport.ControlStructures)

{

Contains = MarkScheme.ControlStructures.Contains(new ExMarkScheme.ControlStructure { StructureType = ConStruct.StructureType, StructureCondition = ConStruct.StructureCondition });

if (MarkScheme.ControlStructures.Contains(new ExMarkScheme.ControlStructure { StructureType = ConStruct.StructureType, StructureCondition = null }))

DNM = true;

else if (MarkScheme.ControlStructures.Contains(new ExMarkScheme.ControlStructure { StructureType = null, StructureCondition = ConStruct.StructureCondition }))

DNM = true;

if (Contains || DNM)

Marks.Add(true);

else

Marks.Add(false);

}

foreach (bool Mark in Marks)

{

if (!Mark)

return false;

}

return true;

}

public static bool MarkOutput()

{

if (MarkScheme.Output == LuaInterpreter.CodeReport.Output)

return true;

else

return false;

}

public static bool MarkVars()

{

if (MarkScheme.AssignedVariables.Contains(new ExMarkScheme.Variable { VarName = null, VarValue = null }))

return true;

bool Contains, DNM = false;

List<bool> Marks = new List<bool>();

foreach (UserCode.Variable Var in LuaInterpreter.CodeReport.AssignedVariables)

{

Contains = MarkScheme.AssignedVariables.Contains(new ExMarkScheme.Variable { VarName = Var.VarName, VarValue = Var.VarValue });

if (MarkScheme.AssignedVariables.Contains(new ExMarkScheme.Variable { VarName = Var.VarName, VarValue = null }))

DNM = true;

else if (MarkScheme.AssignedVariables.Contains(new ExMarkScheme.Variable { VarName = null, VarValue = Var.VarValue }))

DNM = true;

if (Contains || DNM)

Marks.Add(true);

else

Marks.Add(false);

}

foreach (bool Mark in Marks)

{

if (!Mark)

return false;

}

return true;

}

}

}

MarkerLists.cs

using System;

using System.Collections.Generic;

namespace Marker

{

// UserCode and MarkScheme inherit from abstract Code class

public abstract class Code

{

public class Variable : IEquatable<Variable>

{

public string VarName { get; set; }

public string VarValue { get; set; }

public bool Equals(Variable other)

{

throw new NotImplementedException();

}

}

public class ControlStructure : IEquatable<ControlStructure>

{

public string StructureType { get; set; }

public string StructureCondition { get; set; }

public bool Equals(ControlStructure other)

{

throw new NotImplementedException();

}

}

public string Output { get; set; }

public List<Variable> AssignedVariables = new List<Variable>();

public List<ControlStructure> ControlStructures = new List<ControlStructure>();

}

public class ExMarkScheme : Code

{

public bool CheckOutput { get; set; }

public bool CheckVars { get; set; }

public bool CheckConStruct { get; set; }

}

public class UserCode : Code

{

}

}

### Models

Account.cs

using System;

using System.IO;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Text.RegularExpressions;

using System.Web;

namespace Code\_College.Models

{

public static class Account

{

private static UserDBEntities UserDB = new UserDBEntities();

public static void AddCookie(string Username, string Password, HttpResponseBase Response)

{

HttpCookie LoginCookie = new HttpCookie("CCUserAuth");

LoginCookie.Values["Username"] = Username;

LoginCookie.Values["Password"] = Cryptography.Encrypt(Password, Username);

LoginCookie.Expires = DateTime.Now.AddYears(1);

Response.Cookies.Add(LoginCookie);

}

public static void ChangePassword(string Email, string Username, string NewPassword)

{

User user = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (user != null)

{

user.PasswordHash = HashCredentials(Email, NewPassword);

UserDB.SaveChangesAsync();

}

}

public static void CreateNewUser(string Name, string Email, string Username, string Password, HttpResponseBase Response)

{

User user = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (user == null)

{

User NewUser = new User();

NewUser.Name = Name;

NewUser.Email = Email;

NewUser.Username = Username;

NewUser.PasswordHash = HashCredentials(Email, Password);

NewUser.UserLevel = 1;

UserDB.Users.Add(NewUser);

UserDB.SaveChangesAsync();

AddCookie(NewUser.Username, Password, Response);

}

}

public static void DeleteUser(string Username, string Password)

{

User user = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (user != null)

{

UserDB.Users.Remove(user);

UserDB.SaveChangesAsync();

}

}

public static string GetCookieUsername(HttpRequest Request, HttpCookie Cookie)

{

User user = UserDB.Users.Where(x => x.Username == Cookie["Username"]).FirstOrDefault();

if (user == null)

return null;

else if (Cookie["Username"] != null)

return Cookie["Username"];

else

return null;

}

public static string HashCredentials(string Salt, string Password)

{

SHA512 Hash = SHA512.Create();

string Salted = Password + Salt;

string HashedCredentials = Convert.ToBase64String(Hash.ComputeHash(Encoding.UTF8.GetBytes(Salted)));

return HashedCredentials;

}

public static void LevelUp(string Username, Exercise Exercise)

{

User User = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (User.UserLevel <= Exercise.ExID)

{

User.UserLevel++;

UserDB.SaveChangesAsync();

}

}

public static void RemoveCookie(HttpRequest Request)

{

HttpCookie Cookie = Request.Cookies["CCUserAuth"];

if (Cookie != null)

Cookie.Expires.AddYears(-2);

}

public static bool Validation(string Data, char FieldType)

{

bool Validated = false;

switch (FieldType)

{

case 'n':

Regex NameValidator = new Regex("[a-zA-Z'-]", RegexOptions.Compiled);

Validated = NameValidator.IsMatch(Data);

break;

case 'e':

Regex EmailValidator = new Regex("^[a-zA-Z\_0-9]+([-+.'][a-zA-Z\_0-9]+)\*@[a-zA-Z\_0-9]+([-.][a-zA-Z\_0-9]+)\*.[a-zA-Z\_0-9]+([-.][a-zA-Z\_0-9]+)\*$", RegexOptions.Compiled);

Validated = EmailValidator.IsMatch(Data);

break;

case 'u':

Regex UsernameValidator = new Regex("[a-zA-Z0-9'-\_.]", RegexOptions.Compiled);

Validated = UsernameValidator.IsMatch(Data);

break;

}

return Validated;

}

public static bool VerifyCookie(HttpCookie Cookie)

{

string Username = Convert.ToString(Cookie.Values["Username"]);

string Password = Convert.ToString(Cookie.Values["Password"]);

User user = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (user == null)

{

Cookie.Expires.AddYears(-2);

return false;

}

else if (user.PasswordHash == HashCredentials(user.Email, Cryptography.Decrypt(Password, Username)))

return true;

else

{

Cookie.Expires.AddYears(-2);

return false;

}

}

public static bool VerifyUser(string Username, string Password)

{

User user = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (user == null)

return false;

else if (user.PasswordHash == HashCredentials(user.Email, Password))

return true;

else

return false;

}

// That deals with encryption and decryption of password

public static class Cryptography

{

private const int DerivationIterations = 1000;

private const int Keysize = 256;

// Decrypts text

public static string Decrypt(string CipheredText, string PassPhrase)

{

byte[] cipherTextBytesWithSaltAndIv = Convert.FromBase64String(CipheredText);

byte[] saltStringBytes = cipherTextBytesWithSaltAndIv.Take(Keysize / 8).ToArray();

byte[] ivStringBytes = cipherTextBytesWithSaltAndIv.Skip(Keysize / 8).Take(Keysize / 8).ToArray();

byte[] CipherTextBytes = cipherTextBytesWithSaltAndIv.Skip((Keysize / 8) \* 2).Take(cipherTextBytesWithSaltAndIv.Length - ((Keysize / 8) \* 2)).ToArray();

using (Rfc2898DeriveBytes Password = new Rfc2898DeriveBytes(PassPhrase, saltStringBytes, DerivationIterations))

{

byte[] KeyBytes = Password.GetBytes(Keysize / 8);

using (RijndaelManaged SymmetricKey = new RijndaelManaged())

{

SymmetricKey.BlockSize = 256;

SymmetricKey.Mode = CipherMode.CBC;

SymmetricKey.Padding = PaddingMode.PKCS7;

using (var Decryptor = SymmetricKey.CreateDecryptor(KeyBytes, ivStringBytes))

{

using (MemoryStream MemoryStream = new MemoryStream(CipherTextBytes))

{

using (CryptoStream CryptoStream = new CryptoStream(MemoryStream, Decryptor, CryptoStreamMode.Read))

{

var PlainTextBytes = new byte[CipherTextBytes.Length];

var DecryptedByteCount = CryptoStream.Read(PlainTextBytes, 0, PlainTextBytes.Length);

MemoryStream.Close();

CryptoStream.Close();

return Encoding.UTF8.GetString(PlainTextBytes, 0, DecryptedByteCount);

}

}

}

}

}

}

// Encrypts text

public static string Encrypt(string PlainText, string PassPhrase)

{

byte[] SaltStringBytes = Generate256BitsOfRandomEntropy();

byte[] IVStringBytes = Generate256BitsOfRandomEntropy();

byte[] PlainTextBytes = Encoding.UTF8.GetBytes(PlainText);

using (Rfc2898DeriveBytes Password = new Rfc2898DeriveBytes(PassPhrase, SaltStringBytes, DerivationIterations))

{

byte[] KeyBytes = Password.GetBytes(Keysize / 8);

using (RijndaelManaged SymmetricKey = new RijndaelManaged())

{

SymmetricKey.BlockSize = 256;

SymmetricKey.Mode = CipherMode.CBC;

SymmetricKey.Padding = PaddingMode.PKCS7;

using (var Encryptor = SymmetricKey.CreateEncryptor(KeyBytes, IVStringBytes))

{

using (MemoryStream MemoryStream = new MemoryStream())

{

using (CryptoStream CryptoStream = new CryptoStream(MemoryStream, Encryptor, CryptoStreamMode.Write))

{

CryptoStream.Write(PlainTextBytes, 0, PlainTextBytes.Length);

CryptoStream.FlushFinalBlock();

byte[] CipherTextBytes = SaltStringBytes;

CipherTextBytes = CipherTextBytes.Concat(IVStringBytes).ToArray();

CipherTextBytes = CipherTextBytes.Concat(MemoryStream.ToArray()).ToArray();

MemoryStream.Close();

CryptoStream.Close();

return Convert.ToBase64String(CipherTextBytes);

}

}

}

}

}

}

// Generates Random entropy for encryption and decryption

private static byte[] Generate256BitsOfRandomEntropy()

{

byte[] RandomBytes = new byte[32];

using (RNGCryptoServiceProvider RNGCSP = new RNGCryptoServiceProvider())

RNGCSP.GetBytes(RandomBytes);

return RandomBytes;

}

}

}

}

ExDBEntities.cs

using Marker;

using System.ComponentModel.DataAnnotations;

using System.Data.Entity;

namespace Code\_College.Models

{

// Database context for exercise database

public class ExDBEntities : DbContext

{

public ExDBEntities() : base("name=ExDBEntities")

{

}

public virtual DbSet<Exercise> Exercises { get; set; }

}

public class Exercise

{

[Key]

public int DBID { get; set; }

[Required]

public int ExID { get; set; }

[Required]

public string ExTitle { get; set; }

[Required]

public string ExDescription { get; set; }

public string ExCodeTemplate { get; set; }

public string ExAppendCode { get; set; }

public ExMarkScheme ExMarkScheme { get; set; }

}

}

ExerciseParser.cs

using Marker;

using System;

using System.IO;

using System.Xml;

namespace Code\_College.Models

{

public static class ExerciseParser

{

private static ExDBEntities ExDB = new ExDBEntities();

private static StreamReader File;

private static Exercise NewExercise = new Exercise();

private static ExMarkScheme NewMarkScheme = new ExMarkScheme();

public static void ParseExFile(string Filename)

{

// Open exercise file

File = new StreamReader(Filename);

// Parse elements and reset BaseStream position

NewExercise.ExID = ExParsing.GetExID();

File.BaseStream.Position = 0;

NewExercise.ExTitle = ExParsing.GetExTitle();

File.BaseStream.Position = 0;

NewExercise.ExDescription = ExParsing.GetExDescription();

File.BaseStream.Position = 0;

NewExercise.ExCodeTemplate = ExParsing.GetExCodeTemplate();

File.BaseStream.Position = 0;

NewExercise.ExAppendCode = ExParsing.GetExAppendCode();

File.BaseStream.Position = 0;

ExParsing.XMLParsing.ParseXML(ExParsing.GetExMarkSchemeXML());

NewExercise.ExMarkScheme = NewMarkScheme;

ExDB.Exercises.Add(NewExercise);

ExDB.SaveChanges();

}

private static class ExParsing

{

public static string GetExAppendCode()

{

string Entry = "";

while (!File.EndOfStream)

{

string Line = File.ReadLine();

if (Line.StartsWith("[ExAppendCode]"))

{

while (true)

{

Line = File.ReadLine();

if (!Line.StartsWith("[ExMarkScheme]"))

Entry = Entry + " " + Line;

else

return Entry;

}

}

}

return null;

}

public static string GetExCodeTemplate()

{

string Entry = "";

while (!File.EndOfStream)

{

string Line = File.ReadLine();

if (Line.StartsWith("[ExCodeTemplate]"))

{

while (true)

{

Line = File.ReadLine();

if (!Line.StartsWith("[ExAppendCode]"))

Entry = Entry + " " + Line;

else

return Entry;

}

}

}

return null;

}

public static string GetExDescription()

{

string Entry = "";

while (!File.EndOfStream)

{

string Line = File.ReadLine();

if (Line.StartsWith("[ExDescription]"))

{

while (true)

{

Line = File.ReadLine();

if (!Line.StartsWith("[ExCodeTemplate]"))

Entry = Entry + " " + Line;

else

return Entry;

}

}

}

return null;

}

public static int GetExID()

{

while (!File.EndOfStream)

{

string Line = File.ReadLine();

if (!Line.StartsWith("[") && Line != "")

return Convert.ToInt32(Line);

}

return 0;

}

public static XmlDocument GetExMarkSchemeXML()

{

XmlDocument XML = new XmlDocument();

string Entry = "";

string Line = "";

while (!Line.StartsWith("[ExMarkScheme]"))

Line = File.ReadLine();

while (!File.EndOfStream)

{

Line = File.ReadLine();

Entry += Line;

}

if (Entry != "")

XML.LoadXml(Entry);

return XML;

}

public static string GetExTitle()

{

while (!File.EndOfStream)

{

string Line = File.ReadLine();

if (Line.StartsWith("[ExTitle]") && Line != "")

return File.ReadLine();

}

return null;

}

// Class that deals with parsing the XML

public static class XMLParsing

{

public static void ParseXML(XmlDocument XML)

{

NewMarkScheme.Output = GetExOutput(XML);

GetExVars(XML);

GetExConStructs(XML);

}

private static void GetExConStructs(XmlDocument XML)

{

try

{

XmlNodeList ConStructsNode = XML.SelectSingleNode("/MarkScheme/ControlStructures").ChildNodes;

foreach (XmlNode Node in ConStructsNode)

{

NewMarkScheme.CheckConStruct = true;

ExMarkScheme.ControlStructure NewConStruct = new ExMarkScheme.ControlStructure();

NewConStruct.StructureCondition = Node.InnerText;

NewConStruct.StructureType = Node.Attributes.GetNamedItem("StructType").Value;

if (NewConStruct.StructureCondition == "[DNM]")

NewConStruct.StructureCondition = null;

else if (NewConStruct.StructureType == "[DNM]")

NewConStruct.StructureType = null;

NewMarkScheme.ControlStructures.Add(NewConStruct);

}

}

catch

{

ExMarkScheme.ControlStructure NewConStruct = new ExMarkScheme.ControlStructure();

NewMarkScheme.CheckConStruct = false;

NewConStruct.StructureCondition = null;

NewConStruct.StructureType = null;

NewMarkScheme.ControlStructures.Add(NewConStruct);

}

}

private static string GetExOutput(XmlDocument XML)

{

try

{

string Output = XML.SelectSingleNode("/MarkScheme/Output").InnerText;

NewMarkScheme.CheckOutput = true;

return Output;

}

catch

{

NewMarkScheme.CheckOutput = false;

return null;

}

}

private static void GetExVars(XmlDocument XML)

{

try

{

XmlNodeList VariablesNode = XML.SelectSingleNode("/MarkScheme/Variables").ChildNodes;

foreach (XmlNode Node in VariablesNode)

{

NewMarkScheme.CheckVars = true;

ExMarkScheme.Variable NewVar = new ExMarkScheme.Variable();

NewVar.VarName = Node.InnerText;

NewVar.VarValue = Node.Attributes.GetNamedItem("VarValue").Value;

if (NewVar.VarName == "[DNM]")

NewVar.VarName = null;

else if (NewVar.VarValue == "[DNM]")

NewVar.VarValue = null;

NewMarkScheme.AssignedVariables.Add(NewVar);

}

}

catch

{

ExMarkScheme.Variable NewVar = new ExMarkScheme.Variable();

NewMarkScheme.CheckVars = false;

NewVar.VarName = null;

NewVar.VarValue = null;

NewMarkScheme.AssignedVariables.Add(NewVar);

}

}

}

}

}

}

UserDBEntites.cs

using System.ComponentModel.DataAnnotations;

using System.Data.Entity;

namespace Code\_College.Models

{

// Database context for user database

public class User

{

[Key]

public string Name { get; set; }

public string Username { get; set; }

public string Email { get; set; }

public string PasswordHash { get; set; }

public int UserLevel { get; set; }

}

public class UserDBEntities : DbContext

{

public UserDBEntities() : base("name=UserDBEntities")

{

}

public virtual DbSet<User> Users { get; set; }

}

}

### Views

#### AddExercise

Index.cshtml

@using Code\_College.Controllers

@using Code\_College.Models

@{

int Index = 0;

ExDBEntities ExDB = new ExDBEntities();

foreach (Exercise Exercise in ExDB.Exercises)

{

AddExerciseController.ExerciseRows[Index] = new AddExerciseController.ExRow { ExID = Exercise.ExID, ExTitle = Exercise.ExTitle };

Index++;

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta name="keywords" content="Code College, Add Exercise, Learn Programming, Learn Lua, Lua, Development, Code" />

<script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); } </script>

<title>Admin - Add Exercise - Code College</title>

<script type="text/javascript">

var Admin = confirm("This page is for use by admins only.");

if (!Admin) {

window.location = "@Url.Action("Index", "Login")";

}

function Submit() {

document.getElementById("Message").innerHTML = "Submitting file...";

}

</script>

</head>

<body>

<h1>Add Exercise</h1>

<form action="" method="post" enctype="multipart/form-data">

<label for="file">Exercise File:</label>

<input type="file" name="file" id="file" />

<input type="submit" onclick="Submit()" />

</form>

<p id="Message">Upload an exercise.</p>

<div class="Exercises">

<h3>Exercises:</h3>

<ul>

@foreach (AddExerciseController.ExRow ExerciseRow in AddExerciseController.ExerciseRows)

{

string ExRecord = string.Format("{0} | {1}", ExerciseRow.ExID.ToString(), ExerciseRow.ExTitle);

<li>@ExRecord</li>

}

</ul>

</div>

</body>

</html>

#### Dashboard

Index.cshtml

@using Code\_College.Controllers;

@using Code\_College.Models;

@{

bool Redirect = false;

int Index = 0;

ExDBEntities ExDB = new ExDBEntities();

UserDBEntities UserDB = new UserDBEntities();

if (Request.Cookies.AllKeys.Contains("CCUserAuth"))

{

HttpCookie Cookie = Request.Cookies["CCUserAuth"];

if (!Account.VerifyCookie(Cookie))

{

Redirect = true;

}

else

{

string Username = Cookie.Values["Username"];

User User = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

foreach (Exercise Exercise in ExDB.Exercises)

{

if (Exercise.ExID == User.UserLevel)

{

DashboardController.ExerciseTiles[Index] = new DashboardController.ExTile { ExID = Exercise.ExID, ExTitle = Exercise.ExTitle, CompletedByUser = false, Locked = false };

}

else if (Exercise.ExID < User.UserLevel)

{

DashboardController.ExerciseTiles[Index] = new DashboardController.ExTile { ExID = Exercise.ExID, ExTitle = Exercise.ExTitle, CompletedByUser = true, Locked = false };

}

else if (Exercise.ExID > User.UserLevel)

{

DashboardController.ExerciseTiles[Index] = new DashboardController.ExTile { ExID = Exercise.ExID, ExTitle = Exercise.ExTitle, CompletedByUser = false, Locked = true };

}

else

{

DashboardController.ExerciseTiles[Index] = new DashboardController.ExTile { ExID = Exercise.ExID, ExTitle = Exercise.ExTitle, CompletedByUser = false, Locked = true };

}

Index++;

}

}

}

else

{

Redirect = true;

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta name="keywords" content="Code College, Dashboard, Learn Programming, Learn Lua, Lua, Development, Code" />

<script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); } </script> <!----webfonts--->

<link href='https://fonts.googleapis.com/css?family=Roboto' rel='stylesheet' type='text/css'>

<title>Dash - Code College</title>

<style type="text/css">

.DashTitle {

font-family: 'Roboto', sans-serif;

font-size: x-large;

text-align: center;

color: white;

padding-top: 15px;

}

.Exercises {

background-color: white;

padding-bottom: 10px;

}

.Exercise {

text-align: left;

border: thick;

font-size: medium;

padding: 10px;

margin-left: 220px;

margin-right: 220px;

}

.ExerciseTile {

text-align: left;

border: thick;

font-size: medium;

padding: 10px;

padding-bottom: 10px;

}

.Copyright {

color: white;

text-align: center;

font-family: 'Roboto', sans-serif;

font-size: small;

}

</style>

<script type="text/javascript">

var Redirect = "@Redirect";

if (Redirect === "True") {

window.location = "@Url.Action("Index", "Login")";

}

function GotoExercise(ExID) {

window.location = "@Url.Action("Index", "Exercise")?id=" + ExID;

}

</script>

</head>

<body style="background-color: black;">

<div class="DashTitle" style="height: 50px; background-color: #06D995">Code College - Dashboard</div>

<div class="Exercises">

@if (!Redirect)

{

foreach (DashboardController.ExTile Tile in DashboardController.ExerciseTiles)

{

string ExRecord = string.Format("{0} | {1}", Tile.ExID.ToString(), Tile.ExTitle);

if (Tile.CompletedByUser)

{

<div class="Exercise" style="width: 220px; height: 70px; position: relative; left: 315px; background-color: grey;">

<div class="ExerciseTile">

<p>@ExRecord</p>

<button onclick="GotoExercise(@Tile.ExID)">Try Exercise</button>

</div>

</div>

}

if (!Tile.CompletedByUser)

{

if (Tile.Locked)

{

<div class="Exercise" style="width: 220px; height: 70px; position: relative; left: 315px; background-color: white;">

<div class="ExerciseTile">

<p>@ExRecord</p>

<button onclick="GotoExercise(@Tile.ExID)" disabled>Try Exercise</button>

</div>

</div>

}

else

{

<div class="Exercise" style="width: 220px; height: 70px; position: relative; left: 315px; background-color: white;">

<div class="ExerciseTile">

<p>@ExRecord</p>

<button onclick="GotoExercise(@Tile.ExID)">Try Exercise</button>

</div>

</div>

}

}

}

}

</div>

<div class="Copyright">

<p>Copyright © 2015 Code College. All Rights Reserved</p>

</div>

</body>

</html>

#### Exercise

Index.cshtml

@using System.Web.Optimization;

@using Code\_College.Models;

@{

bool Redirect = false;

int ExerciseID = ViewBag.ExID;

UserDBEntities UserDB = new UserDBEntities();

if (Request.Cookies.AllKeys.Contains("CCUserAuth"))

{

HttpCookie Cookie = Request.Cookies["CCUserAuth"];

if (!Account.VerifyCookie(Cookie))

{

Redirect = true;

}

else

{

string Username = Cookie.Values["Username"];

string Password = Cookie.Values["Password"];

ViewBag.Username = Username;

ViewBag.Password = Password;

User User = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (ExerciseID > User.UserLevel)

{

Redirect = true;

}

}

}

else

{

Redirect = true;

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta name="keywords" content="Code College, Exercise, Learn Programming, Learn Lua, Lua, Development, Code, @ViewBag.ExerciseTitle" />

<script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); } </script>

<title>@ViewBag.Title</title>

@Styles.Render("~/bundles/editorcss")

@Scripts.Render("~/bundles/editorjs")

@Scripts.Render("~/bundles/js")

<style>

#Console {

background: #1A1A1A;

height: 30px;

width: 300px;

border-radius: 2em;

border: none;

color: #A2A2A2;

padding-left: 1.5em;

outline: none;

box-shadow: 0 4px 6px -5px hsl(0, 0%, 40%), inset 0px 4px 6px -5px hsl(0, 0%, 2%);

}

.ExTitle {

font-family: 'Roboto', sans-serif;

font-size: x-large;

text-align: center;

color: white;

padding-top: 15px;

}

.ExDesc {

font-family: Arial;

color: #FFFFFF;

padding-top: 10px;

padding-bottom: 30px;

}

.Message {

color: white;

}

.Copyright {

color: white;

text-align: center;

font-family: 'Roboto', sans-serif;

font-size: small;

}

</style>

<script>

var Redirect = "@Redirect";

if (Redirect === "True") {

window.location = "@Url.Action("Index", "Login")";

}

function UpdateConsole()

{

var Submission = new Object();

Submission.id = @ViewBag.ExerciseID;

Submission.username = "@ViewBag.Username";

Submission.password = "@ViewBag.Password";

Submission.code = editor.getValue() + "@ViewBag.AppendCode";

$.ajax({

url: '/api/code',

type: 'POST',

dataType: 'json',

data: Submission,

success: function (data, textStatus, xhr) {

console.log(data);

document.getElementById("Console").value = data;

CodeCorrect(data);

},

error: function (xhr, textStatus, errorThrown) {

console.log('Error in Operation');

console.log(xhr);

console.log(textStatus);

console.log(errorThrown);

}

});

}

function CodeCorrect (ConsoleOutput)

{

if (ConsoleOutput === "Sorry, that was incorrect. Please, read the task and try again.") {

document.getElementsByClassName("Message")[0].innerHTML = "Incorrect, your code does not adhere to the task set, sorry.";

document.getElementById("Console").style['background'] = 'red';

}

else if (ConsoleOutput.startsWith('Interpreter:')) {

document.getElementsByClassName("Message")[0].innerHTML = "Incorrect, you have a syntax error in your code.";

document.getElementById("Console").style['background'] = 'red';

}

else {

document.getElementsByClassName("Message")[0].innerHTML = "Correct!";

document.getElementById("button").onclick = NextExercise;

document.getElementById("button").innerHTML = "Next Exercise";

document.getElementById("Console").style['background'] = 'green';

}

}

function NextExercise() {

var ExerciseID = @ViewBag.ExID;

ExerciseID++;

window.location = "@Url.Action("Index", "Exercise")?id=" + ExerciseID;

}

</script>

</head>

<body style="background-color: black;">

<div class="ExTitle" style="height: 50px; background-color: #06D995">@ViewBag.ExerciseTitle</div>

<div class="ExDesc">

@ViewBag.Desc

</div>

<div>

<textarea id="Code" name="code" value="" style="width: 300px; height: 65px"></textarea>

<button id="button" onclick="UpdateConsole()">Submit</button>

</div>

<div>

Console:

<br />@Html.TextBox("Console", null, new { id = "Console" })<br />

</div>

<div class="Message"></div>

<script>

var CodeTemplate = "@ViewBag.CodeTemplate";

var Temp

var editor = CodeMirror.fromTextArea(document.getElementById("Code"), {

mode: "lua",

lineNumbers: true,

matchBrackets: true,

theme: "night"

});

while (CodeTemplate != Temp) {

Temp = CodeTemplate;

CodeTemplate = CodeTemplate.replace("&quot;", "\"");

}

editor.getDoc().setValue(CodeTemplate);

</script>

<div class="Copyright">

<p>Copyright © 2015 Code College. All Rights Reserved</p>

</div>

</body>

</html>

#### Login

Index.cshtml

@using System.Web.Optimization;

@using Code\_College.Models;

@{

bool Redirect = false;

int ExerciseID = ViewBag.ExID;

UserDBEntities UserDB = new UserDBEntities();

if (Request.Cookies.AllKeys.Contains("CCUserAuth"))

{

HttpCookie Cookie = Request.Cookies["CCUserAuth"];

if (!Account.VerifyCookie(Cookie))

{

Redirect = true;

}

else

{

string Username = Cookie.Values["Username"];

string Password = Cookie.Values["Password"];

ViewBag.Username = Username;

ViewBag.Password = Password;

User User = UserDB.Users.Where(x => x.Username == Username).FirstOrDefault();

if (ExerciseID > User.UserLevel)

{

Redirect = true;

}

}

}

else

{

Redirect = true;

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta name="keywords" content="Code College, Exercise, Learn Programming, Learn Lua, Lua, Development, Code, @ViewBag.ExerciseTitle" />

<script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); } </script>

<title>@ViewBag.Title</title>

@Styles.Render("~/bundles/editorcss")

@Scripts.Render("~/bundles/editorjs")

@Scripts.Render("~/bundles/js")

<style>

#Console {

background: #1A1A1A;

height: 30px;

width: 300px;

border-radius: 2em;

border: none;

color: #A2A2A2;

padding-left: 1.5em;

outline: none;

box-shadow: 0 4px 6px -5px hsl(0, 0%, 40%), inset 0px 4px 6px -5px hsl(0, 0%, 2%);

}

.ExTitle {

font-family: 'Roboto', sans-serif;

font-size: x-large;

text-align: center;

color: white;

padding-top: 15px;

}

.ExDesc {

font-family: Arial;

color: #FFFFFF;

padding-top: 10px;

padding-bottom: 30px;

}

.Message {

color: white;

}

.Copyright {

color: white;

text-align: center;

font-family: 'Roboto', sans-serif;

font-size: small;

}

</style>

<script>

var Redirect = "@Redirect";

if (Redirect === "True") {

window.location = "@Url.Action("Index", "Login")";

}

function UpdateConsole()

{

var Submission = new Object();

Submission.id = @ViewBag.ExerciseID;

Submission.username = "@ViewBag.Username";

Submission.password = "@ViewBag.Password";

Submission.code = editor.getValue() + "@ViewBag.AppendCode";

$.ajax({

url: '/api/code',

type: 'POST',

dataType: 'json',

data: Submission,

success: function (data, textStatus, xhr) {

console.log(data);

document.getElementById("Console").value = data;

CodeCorrect(data);

},

error: function (xhr, textStatus, errorThrown) {

console.log('Error in Operation');

console.log(xhr);

console.log(textStatus);

console.log(errorThrown);

}

});

}

function CodeCorrect (ConsoleOutput)

{

if (ConsoleOutput === "Sorry, that was incorrect. Please, read the task and try again.") {

document.getElementsByClassName("Message")[0].innerHTML = "Incorrect, your code does not adhere to the task set, sorry.";

document.getElementById("Console").style['background'] = 'red';

}

else if (ConsoleOutput.startsWith('Interpreter:')) {

document.getElementsByClassName("Message")[0].innerHTML = "Incorrect, you have a syntax error in your code.";

document.getElementById("Console").style['background'] = 'red';

}

else {

document.getElementsByClassName("Message")[0].innerHTML = "Correct!";

document.getElementById("button").onclick = NextExercise;

document.getElementById("button").innerHTML = "Next Exercise";

document.getElementById("Console").style['background'] = 'green';

}

}

function NextExercise() {

var ExerciseID = @ViewBag.ExID;

ExerciseID++;

window.location = "@Url.Action("Index", "Exercise")?id=" + ExerciseID;

}

</script>

</head>

<body style="background-color: black;">

<div class="ExTitle" style="height: 50px; background-color: #06D995">@ViewBag.ExerciseTitle</div>

<div class="ExDesc">

@ViewBag.Desc

</div>

<div>

<textarea id="Code" name="code" value="" style="width: 300px; height: 65px"></textarea>

<button id="button" onclick="UpdateConsole()">Submit</button>

</div>

<div>

Console:

<br />@Html.TextBox("Console", null, new { id = "Console" })<br />

</div>

<div class="Message"></div>

<script>

var CodeTemplate = "@ViewBag.CodeTemplate";

var Temp

var editor = CodeMirror.fromTextArea(document.getElementById("Code"), {

mode: "lua",

lineNumbers: true,

matchBrackets: true,

theme: "night"

});

while (CodeTemplate != Temp) {

Temp = CodeTemplate;

CodeTemplate = CodeTemplate.replace("&quot;", "\"");

}

editor.getDoc().setValue(CodeTemplate);

</script>

<div class="Copyright">

<p>Copyright © 2015 Code College. All Rights Reserved</p>

</div>

</body>

</html>

#### SignUp

Index.cshtml

@using System.Web.Optimization;

@using Code\_College.Models;

@{

bool Redirect = false;

if (Request.Cookies.AllKeys.Contains("CCUserAuth"))

{

HttpCookie Cookie = Request.Cookies["CCUserAuth"];

if (Account.VerifyCookie(Cookie))

{

Redirect = true;

}

}

if (IsPost)

{

if (Request.Form["Password"] == Request.Form["RetypePassword"])

{

string Name = Request.Form["Name"];

string Email = Request.Form["Email"];

string Username = Request.Form["Username"];

string Password = Account.HashCredentials(Username, Request.Form["Password"]);

if (Account.Validation(Name, 'n') && Account.Validation(Email, 'e') && Account.Validation(Username, 'u'))

{

Account.CreateNewUser(Name, Email, Username, Password, Response);

Redirect = true;

}

}

}

}

<!DOCTYPE html>

<html>

<head>

<meta name="viewport" content="width=device-width, initial-scale=1">

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<meta name="keywords" content="Code College, Sign Up, Learn Programming, Learn Lua, Lua, Development, Code" />

<script type="application/x-javascript"> addEventListener("load", function() { setTimeout(hideURLbar, 0); }, false); function hideURLbar(){ window.scrollTo(0,1); } </script>

<title>Sign Up - Code College</title>

@Styles.Render("~/bundles/css")

@Scripts.Render("~/bundles/js")

<script>

var Redirect = "@Redirect";

if (Redirect === "True") {

window.location = "@Url.Action("Index", "Dashboard")";

}

</script>

</head>

<body id="login">

<div class="login-logo">

<a href="http://ryankendrick.co.uk/"><img src="~/Images/logo.png" alt="" /></a>

</div>

<h2 class="form-heading">Register</h2>

<form id="signupForm" class="form-signin app-cam" action="/SignUp" method="post">

<p>Enter your personal details below:</p>

@Html.TextBox("Name", null, new { placeholder = "Name" })

@Html.TextBox("Email", null, new { placeholder = "Email" })

<p>Enter your account details below:</p>

@Html.TextBox("Username", null, new { placeholder = "Username" })

@Html.Password("Password", null, new { placeholder = "Password" })

@Html.Password("RetypePassword", null, new { placeholder = "Re-Type Password" })

<button class="btn btn-lg btn-success1 btn-block" type="submit">Submit</button>

<div class="registration">

Already Registered?

<a class="" href="/Login">

Login

</a>

</div>

</form>

<div class="copy\_layout login register">

<p>Copyright &copy; 2015 Code College. All Rights Reserved | Design by <a href="http://w3layouts.com/" target="\_blank">W3layouts</a> </p>

</div>

</body>

</html>