LearningEDU

ChessEDU
Introduction

Version <2.1>

Revision History

Date	Version	Description	Author
25/10/2022	1.0	First Draft	Adair Torres
30/10/2022	2.1	Reformatted document to match Teaching Assistant's specifications.	Adair Torres

Table of Contents

1. Product Vision 4

2. Profiles 4

3. Role and Responsibilities 6

Introduction

1. Product Vision

FOR enthusiastic players of chess all the way from brand new to intermediate levels WHO wish to learn the game at their own pace incrementally, THE ChessEDU App is an educational service THAT offers small, accessible, and interactive lessons that teach users the basics and the very heart of the game. UNLIKE other chess products, services, or software applications, such as Chess.com, the focus of ChessEDU is bite-sized interactive lessons, rather than long and complex modules paired with practice, OUR PRODUCT provides an easy way to learn this intimidating game through not just theory, but practice using lessons that give the user strategy piece by piece and present opportunities to apply what they have learned immediately.

2. Profiles

Name: Grant Jones

Phone Number: (913)645-4050

Email: g641j712@ku.edu

Available times for team meetings: MWF 8am-1pm, TR 8am-2:30pm

Major: Interdisciplinary Computing with an Emphasis in Biology

Year: Junior

Relevant Courses: EECS 168, 268, 368

Proficient Programming Languages: C++, JavaScript, Java, and HTML

Hobbies: Watching and playing sports

Name: Joe Murray

Phone Number: 913-269-0760

Email: j604m256@ku.edu

Available times for team meetings: MW 11-1, TuesThur 9:30-10:30, 12-2

Major: Computer Science

Year: Senior

Relevant Courses: EECS 168,268,368,

Proficient Programming Languages: C++, javascript, python

Hobbies: Basketball, bowling, martial arts

Name: Adair Torres

Phone Number: (620) 640-7414

Email: adair.tor24@ku.edu

Available times for team meetings: M: 10am-10pm, TuTh: 4pm-10pm, W: 4pm-10pm, F: 6pm-10pm

Major: Computer Science

Year: Junior

Relevant Courses: EECS 168, EECS 268, EECS 368

Proficient Programming Languages: C++, JavaScript, C#, Python, HTML

Hobbies: Tabletop & digital games, digital art, weightlifting, racquetball

Name: Jack Reynolds

Phone Number: (913)-634-0412

Email: jackreynolds@ku.edu

Available times for team meetings: M: 8pm - 10 pm, TWRF: 6pm - 10pm, flexible Saturday/Sunday

Major: Computer Science

Year: Junior

Relevant Courses: EECS 168, EECS 268, EECS 368

Proficient Programming Languages: C++, C, JavaScript, HTML

Hobbies: Playing saxophone, tabletop and video games, cooking

Name: Rylan DeGarmo

Phone Number: (316) 796-3719

Email: r031d544@ku.edu

Available times for team meetings: M/F 8am-9am, 4pm-10pm | Tu/Th 8am-10am, 4pm-10pm | W 8am-9am, 12pm-10pm | Sa/Su 8am-10pm

Major: Computer Science

Year: Junior

Relevant Courses: EECS 168, EECS 268, EECS 140, EECS 210

Proficient Programming Languages: C++, HTML

Hobbies: Youtube, video games

Name: Chinh Nguyen

Phone Number: (620) 277-6337

Email: nguyenchinh@ku.edu

Available times for team meetings: M-W-F 5PM-10PM, Flexible Saturday and Sunday

Major: Computer Science

Year: Junior

Relevant Courses: EECS 168, EECS 268, EECS 368

Proficient Programming Languages: Python, JavaScript, C++, HTML

Hobbies: Video games, weight lifting, art, music

3. Role and Responsibilities

Name: Grant Jones

Role: Project Manager

Responsibilities: Responsible for providing up-to-date status of team progress, managing the team meetings, and maintaining a record (minutes or log) of each meeting (e.g., when, purpose, who attended, etc.).

Name: Chinh Nguyen

Role: Quality Assurance Engineer

Responsibilities: Responsible for the final quality of each artifact, e.g., technical accuracy, but also uniformity in typesetting (consistency of font sizes, margins, colors), correct spelling and grammatically correct sentences, adhering to the templates, checking for consistency among deliverables, etc.

Name: Rylan DeGarmo

Role: Project Leader

Responsibilities: Responsible for compiling “original project deliverables” which has been accomplished by all team members, directing the project and leading project portion meetings, and reporting to the professor project technical issues not resolvable within the team.

Name: Joe Murray

Role: ABSENT

Responsibilities: N/A

Name: Adair Torres

Role: Data Administrator

Responsibilities: Responsible for development of file server system backend, which may track user lessons, store login credentials, and organize and deliver files required by individual lessons.

Name: Jack Reynolds

Role: UI / Accessibility Developer

Responsibilities: Responsible for the development of frontend software, user interfaces and displays. Emphasis on creating user assets that can be used without significant difficulty.

4. Meeting Log

Date	Time	Description	Attendance
10/9/22	75 minutes	Fill out profiles, discuss roles, and brainstorm project ideas	Adair, Jack, Rylan, Chinh, and Grant
17/9/22	40 minutes	Decide what project we're doing and create a product vision statement	Jack, Rylan, Chinh, Joe, Adair, and Grant
28/9/22	85 minutes	Discussing outline for use case requirements	Jack, Rylan (small group)
1/10/22	35 minutes	Finalizing first iteration of requirements, clarification on individual work	Adair, Jack, Rylan, Chinh
8/10/22	35 minutes	Finalized requirements, prototype, and GitHub. Discussed next steps	Adair, Jack, Chinh, Rylan, and Grant
22/10/22	30 minutes	Discuss and distribute work for first iteration of UML class and sequence diagrams	Adair, Jack, Rylan, Chinh, and Grant
22/10/22	50 minutes	Created first iteration of UML class diagrams	Rylan and Grant (small group)
26/10/22	30 minutes	Organized and discussed work for Software Architecture Document.	Adair, Jack, Chinh, Rylan, and Grant
10/11/22	30 minutes	Discussed finalization of Iteration 1, and organized workload to begin Iteration 2.	Adair, Jack, Rylan, and Grant
17/11/22	40 minutes	Discussed finalizing Iteration 3 to implement course browser and progress tracking.	Adair, Jack, Rylan, Chinh, and Grant

LearningEDU

ChessEDU
Glossary

Version <2.2>

Revision History

Date	Version	Description	Author
25/10/2022	1.0	First Draft	Adair Torres
28/10/2022	2.0	Second Draft	Rylan DeGarmo
29/10/2022	2.1	Updated formatting to fit template and numbering Minor revisions to wording for understandability Alphabetized lists of terminologies	Chinh Nguyen
30/10/2022	2.2	Updated table of contents.	Adair Torres

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Scope 4
   - 1.3 References 4
   - 1.4 Overview 4

2. Definitions 5
   - 2.1 ChessEDU General Terminology 5
     - 2.1.1 Lesson 5
     - 2.1.2 Module 5
     - 2.1.3 Registered User 5
     - 2.1.4 Unregistered User 5
   - 2.2 Chess-Related Terminology 5
     - 2.2.1 Activate / Activated 5
     - 2.2.2 Attacked / Threatened / Under Attack 5
     - 2.2.3 Castling 5
     - 2.2.4 Check 5
     - 2.2.5 Checkmate 5
     - 2.2.6 Defended 5
     - 2.2.7 Discovered 5
     - 2.2.8 En Passant [French for “in passing”] 5
     - 2.2.9 Forced 6
     - 2.2.10 Fork / Forking 6
     - 2.2.11 Material 6
     - 2.2.12 Pin / Pinning 6
     - 2.2.13 Skewer / Skewering 6
     - 2.2.14 Stalemate 6
     - 2.2.15 Trade 6

Glossary

1. Introduction

1.1 Purpose

The Glossary contains a list of all terminology used in the ChessEDU project to help facilitate understanding of terms that may be unknown or otherwise have special meanings in context of this product such that they may be understood in full.

1.2 Scope

This glossary contains a list of all terminology used in the project in relation to what the project means by them. Use cases, actors, and similar terminology is handled in the sections specified for them for ease of understanding, and as such will not be present in this section.

1.3 References

No additional references are needed.

1.4 Overview

The rest of this document contains definitions regarding the technical and backend of the project as well as specific terminology related to chess, a popular boardgame.

2. Definitions

2.1 ChessEDU General Terminology

2.1.1 Lesson

Short interactive experience for the user to learn something new about chess

2.1.2 Module

A group of lessons

2.1.3 Registered User

A user who does have an account

2.1.4 Unregistered User

A user who does not have an account

2.2 Chess-Related Terminology

2.2.1 Activate / Activated

A player's piece is activated when it is now able to move freely or attack opposing pieces after not being able to do so before. This happens when friendly pieces that were restricting this piece's movement are moved out of the way. The rook, as an example, starts the game inactive since it cannot be used to any real benefit without multiple turns of moving the pieces that surround it.

2.2.2 Attacked / Threatened / Under Attack

A player's piece could be taken by one of their opponent's pieces during the opponent's next turn.

2.2.3 Castling

A special move between a player's King and one of their Rooks that shifts around the positions of both pieces at once.

2.2.4 Check

A state of the game where one player's King is under attack by one of their opponent's pieces. The player in check must eliminate the threat to their King during their turn by moving out of the way, blocking with another piece, or capturing the piece that caused the check.

2.2.5 Checkmate

A state of the game where one player's King is under attack, but they have no move they can take to secure their King again. This is a winning state for the opposing player.

2.2.6 Defended

A player's piece is defended if that piece being captured would open the attacking opponent piece up to being captured. The opponent cannot take this piece without losing one of their own.

2.2.7 Discovered

A discovered attack is a move by one player's piece that opens a new angle of attack for a different one of that same player's pieces.

2.2.8 En Passant [French for "in passing"]

A special pawn move that can only be used against an opponent's pawn that just moved two spaces at once. If the opposing pawn could have been taken by the player's pawn if it had moved one space that turn instead of two, the player can take this pawn and move to the space where the opposing pawn would have been if it had moved only one space.

2.2.9 Forced

A player, during their turn, has only one option of move to make that follows the rules. Occurs most often when their King is in check.

2.2.10 Fork / Forking

A strategy where an opponent guarantees a gain in material by threatening two pieces at once. The player being attacked can only move one piece out of danger and must give up the other.

2.2.11 Material

A representation of the strategic value of a piece. A piece worth higher material is more useful than a piece of lower material in almost all cases. This concept is often used to quantify which player is currently "winning" or "losing." A player is considered "up in material" if the total value of pieces they have taken from their opponent is greater than the total value of pieces lost to their opponent.

2.2.12 Pin / Pinning

A strategy where an opponent threatens a high material piece by threatening a lower value piece on the same line of attack. The lower value piece cannot be moved without opening up the higher value piece to be captured by the opponent on the next turn.

2.2.13 Skewer / Skewering

A strategy where an opponent gains material by threatening a high value piece that is on the same line of attack as a lower value piece. The threatened player, to avoid losing a valuable piece, moves the higher value piece out of danger and gives up the lower value piece in the process.

2.2.14 Stalemate

A state of the game where the player up to move has no valid moves. This results in the game ending in a draw.

2.2.15 Trade

A sequence of moves where a player removes pieces of some material value in exchange for an equal amount of material cost to do so.

LearningEDU

ChessEDU
Software Requirements Specifications

Version <2.1>

Revision History

Date	Version	Description	Author
25/10/2022	1.0	First Draft	Adair Torres
26/10/2022	2.0	Second Draft	Grant Jones
30/10/2022	2.1	Edited for consistent formatting and spacing, updated table of contents.	Adair Torres

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Scope 4
   - 1.3 Definitions, Acronyms, and Abbreviations 4
   - 1.4 References 4
   - 1.5 Overview 4

2. Overall Description 5
   - 2.1 Product perspective 5
     - 2.1.1 System Interfaces 5
     - 2.1.2 User Interfaces 5
     - 2.1.3 Hardware Interfaces 5
     - 2.1.4 Software Interfaces 5
     - 2.1.5 Communication Interfaces 5
     - 2.1.6 Memory Constraints 6
     - 2.1.7 Operations 6
   - 2.2 Product functions 6
   - 2.3 User characteristics 6
   - 2.4 Constraints 6
   - 2.5 Assumptions and dependencies 6
   - 2.6 Requirements subsets 6

3. Specific Requirements 7
   - 3.1 Functionality 7
     - 3.1.1 Lessons 7
     - 3.1.2 Settings 7
     - 3.1.3 Playing the Game 7
     - 3.1.4 Home Screen 7
   - 3.2 Use-Case Specifications 8
     - 3.2.1 Account Creation 8
     - 3.2.2 Signing In 8
     - 3.2.3 Signing Out 8
     - 3.2.4 Selecting a Lesson 8
     - 3.2.5 Playing on Your Own 9
     - 3.2.6 Editing User Settings 9
   - 3.3 Supplementary Requirements 10
     - 3.3.1 Client / Server 10
     - 3.3.2 Security 10
     - 3.3.3 Accessibility 10

4. Classification of Functional Requirements 11

5. Appendixes 14

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this document is to define and describe the software requirements for a web application that will teach the users the basics of chess.

The intended use of this document is for software developers of the web application.

1.2 Scope

The chess app, ChessEDU, using interactive lessons will teach users how to move each piece, various attacking patterns, checks, defending, pins and skewers, two main line openings for white and black, and en passant.

ChessEDU could be used to teach anyone to play chess with the use of the lesson modules. Users of ChessEDU will be able to showcase their knowledge of the game by playing against a friend with the app in a pass and play format. The objective of ChessEDU is to make it easier to learn the intimidating game of chess. Beginner players of chess use ChessEDU to learn the basics of chess like individual piece movement, attacking patterns, and checks. Intermediate players of chess use ChessEDU to learn more about advanced aspects of chess like defending, pins and skewers, openings, and en passant. Advanced players of chess use ChessEDU as a review of the basic and advanced concepts of chess.

1.3 Definitions, Acronyms, and Abbreviations

Chapter:	Groups of lessons
GM:	Gameplay Module
LM:	Lesson Module
SM:	Server Module
SRS:	Software Requirements Specifications

1.4 References

No references.

1.5 Overview

The rest of this document contains a succinct description of the ChessEDU software system (Section 2), and the software requirements specifications for the system (Section 3).

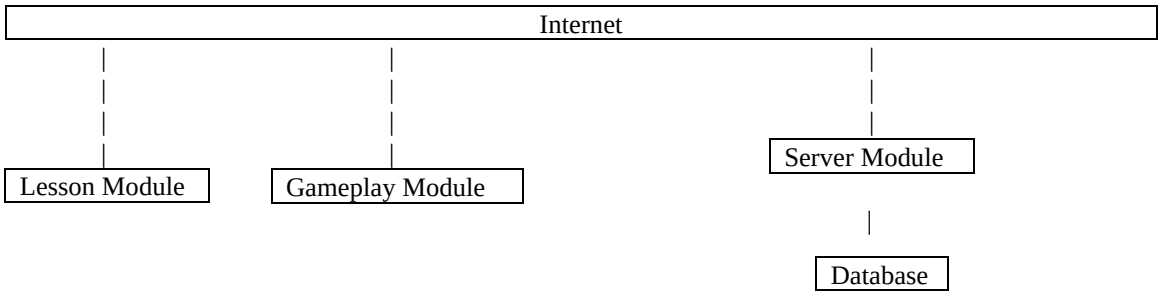
2. Overall Description

2.1 Product perspective

Chess can be a very intimidating game to learn, and thec hess population is growing after the release of the Queen's Gambit on Netflix. Chess is a fun game and has alternative benefits like the development of higher-level thinking. Without an easy way to learn the game, many people may feel discouraged to learn chess because of its complicated nature. A chess tutorial app can solve this problem.

2.1.1 System Interfaces

The ChessEDU system is to be developed is a stand-alone tool that can be accessed through the Internet. It consists of in four major components: a Server Module, a Database, a Lesson Module, a Gameplay Module.



The LM allows users to log onto the ChessEDU system, so users are able to track their progress of the lessons they've completed. The SM allows the LM and GM to connect to it and is as an interface between the modules and the database. The GM allows a user to play a game of chess in a pass and play format. The database can be any type of database and doesn't have to be developed within the ChessEDU system, provided that the SM can interact with the available database system. All components must execute on Windows.

2.1.2 User Interfaces

The LM and GM must provide a user interface that is utilized through Flask. The SM must be able to launch on command but doesn't require a user interface. The database will not have a user interface.

2.1.3 Hardware Interfaces

All components must be able to perform on a personal computer.

2.1.4 Software Interfaces

The LM and GM must be Python scripts running within Flask. The SM must run within a web server available for Windows.

2.1.5 Communication Interfaces

The LM and GM will communicate with the server over a TCP/IP connection. The SM and database will be located on the same host.

2.1.6 Memory Constraints

The LM and GM must be able to operate within 64MB. The SM and database must be able to operate within 128MB.

2.1.7 Operations

The operation of the LM and GM must be easy and intuitive for users. No specific technology knowledge or experience should be required to use the app. The SM will be installed and maintained with no interaction with existing software and not require any technical skills from the network administrator.

Backup operations will be defined.

Recovery operations will be defined in case of network failure, user machine failure, and database failure.

2.2 Product functions

The two main functions of ChessEDU are to teach users basic and advanced concepts of chess and to allow users to practice their new skills in a pass and play game format.

A lesson will teach the user a concept of chess and then ask the user to apply the newly acquired knowledge in a puzzle. Once the user has completed all the lessons for the given chapter, the user will have completed the chapter and be able to move onto the next chapter.

The user will also be able to showcase their knowledge with their friends in a pass and play format where the users will take turns moving their respective pieces on the same device.

The database stores user configurations, which consists of settings, email, username, and password.

2.3 User characteristics

Users are primarily chess beginners and intermediates. Users want to learn how to play the game of chess through interactive modules.

2.4 Constraints

The system will have user authentication security.

2.5 Assumptions and dependencies

No specific assumptions or dependencies.

2.6 Requirements subsets

None.

3. Specific Requirements

3.1 Functionality

3.1.1 Lessons

3.1.1.1 Each lesson will be interactive, meaning each lesson will have at least have one way for the user to apply or demonstrate the acquired knowledge.

3.1.1.2 Each chapter ends with a cumulative review.

3.1.2 Settings

3.1.2.1 User can toggle the following settings: highlighting possible moves after selecting a piece, highlighting of hanging pieces, highlighting of pinned pieces, highlighting of pieces in or entering capture spaces, notification when a player is in check, confirm each move, Auto-Queen when pawn promotes, highlighting last move, sounds, and change theme.

3.1.3 Playing the Game

3.1.3.1 A pair of users can play a game on a single device.

3.1.3.2 During the game, the user has the option to resign, offer draw, and see previous moves.

3.1.3.3 If a player offers a draw, the other player will have the option to accept or decline the draw.

3.1.3.4 If the player accepts the draw, then a popup will state “Draw” with the option to play again or go back to the home screen.

3.1.3.5 If the player declines the draw, then the other player will be notified that the player declined the draw, and the game will resume.

3.1.3.6 After checkmate, resignation, or stalemate, a popup will state “Black wins”, “White wins”, or “Draw” with the option to play again or go back to home screen.

3.1.4 Home Screen

3.1.4.1 Home screen will have the option to play a game or go to lessons.

3.1.4.2 If the user chooses to play a game, then the user will be provided a board to play with a friend locally.

3.1.4.3 If the user chooses to go to lessons, then the next recommended lesson will pop up with the remaining lessons underneath.

3.2 Use-Case Specifications

3.2.1 Account Creation

3.2.1.1 Actor is an unregistered user.

3.2.1.2 Server needs to request email, username, and password.

3.2.1.3 A user account with this information is created in the database afterwards.

3.2.1.4 If the login is already taken, then the process will loop to prevent the actor from progressing until unique account information is entered.

3.2.1.5 If the actor is already signed in, then this specific actor should not be allowed to enter the system and prompt them if they want to sign out instead.

3.2.2 Signing In

3.2.2.1 Actor is a registered user that is not logged in.

3.2.2.2 The actor needs to be prompted with a login page.

3.2.2.3 Login page needs to contact the database to validate the user.

3.2.2.4 Actor should be redirected to a menu with their account specific features.

3.2.2.5 If the actor is signed in, then the specific actor should not be allowed to enter this system and the actor will be prompted if they want to sign out instead.

3.2.2.6 If the actor enters invalid information, the interface will prevent the actor from progressing and needs to prompt them about what is invalid (ex: wrong password).

3.2.3 Signing Out

3.2.3.1 Actor is a registered user that is logged in.

3.2.3.2 Any other user should not be able to see the option to sign out.

3.2.3.3 Server will be contacted that the user wants to sign out.

3.2.3.4 Actor is redirected to the view of the page a signed-out user can see.

3.2.4 Selecting a Lesson

3.2.4.1 Actor is any user (unregistered/registered)

3.2.4.2 Actor needs access to a catalog of available lessons

3.2.4.3 Server needs to gather lessons from the database that contains them, handling any connection issues in the process

3.2.4.4 Actor can choose to enter a lesson, which hands control to a new display

3.2.4.5 As a lesson progresses, the interface includes different options displayed to the user: a user can flip from one page to the next/previous page, interface accesses pages from the server database, pages

need to display text to the user in an easily readable manner, interactive elements (such as small chess boards).

3.2.4.6 The interactive elements includes: the user can move pieces in real time, element provides feedback (ex: “good move!”) after an action, element accurately updates the position of pieces, a piece can be “locked” so the user cannot move it

3.2.4.7 Actor can leave the lesson at any point and save progress if logged in

3.2.5 Playing on Your Own

3.2.5.1 Actor is any user (unregistered/registered)

3.2.5.2 Actor can leave this page at any time

3.2.5.3 Control alternates between a “white” and a “black” player (white goes first).

3.2.5.4 The other player’s pieces are locked during one player’s turn.

3.2.5.5 Player is prompted that it is their move.

3.2.5.6 Player can move a piece to a position.

3.2.5.7 If the movement is invalid for that piece: do not update the board, prompt the player that the move is invalid, and give the player the ability to move again.

3.2.5.8 If the movement is valid for that piece: update the board.

3.2.5.9 If the movement captures a piece: the piece is removed from the board.

3.2.5.10 Control shifts to the opposing player when the current player makes a valid move

3.2.5.11 If the player is in “check”: a different set of movement rules should be applied so that the user has to address the threat.

3.2.5.12 The interface should check if the board state after a move produces either a checkmate or stalemate.

3.2.5.13 If there’s a checkmate, the player who last moved is the “winner” of the game

3.2.5.14 If there’s a stalemate, the interface should prompt the players that there is a draw

3.2.6 Editing User Settings

3.2.6.1 Actor is a registered user.

3.2.6.2 All other actors should not have access to this menu.

3.2.6.3 The actor should be prompted with a list of settings.

3.2.6.4 When the actor updates a setting, the server should receive the request and save the change to the database.

3.2.6.5 If a setting is sensitive (“change password”, “update email”): the actor should be prompted to

validate their information.

3.2.6.6 After three attempts of validating their information, they should be signed out

3.2.6.7 Email confirmation for specific setting changes like password will be sent.

3.3 Supplementary Requirements

3.3.1 Client / Server

3.3.1.1 The user interacts with the system through a mobile or a web browser

3.3.1.2 Either will execute HTML and Java queries to the web engine using JSP

3.3.1.3 The engine communicates with the database using SQL

3.3.1.4 The database returns the data to the engine which, in turn, returns the resulting package to the browser or application, displaying the results afterwards.

3.3.2 Security

3.3.2.1 Individual users have the option to create accounts within the system for tracking personal progress and data.

3.3.2.2 Account credentials will need to be stored within the database, made publicly inaccessible, and encrypted to guarantee security.

3.3.2.3 No information beyond bare minimum identification and module progress will be stored to minimize the impact of a data leakage should security measures fail.

3.3.3 Accessibility

3.3.3.1 System should include accessibility the certain groups of users depend on in order to utilize any software, including but not limited to: text narration, screen readers, variable font size, audio captions, and color correction.

4. Classification of Functional Requirements

Functionality	Type
Each lesson will be interactive, meaning each lesson will have at least have one way for the user to apply or demonstrate the acquired knowledge.	Desirable
Each chapter ends with a cumulative review.	Optional
User can toggle the following settings: highlighting possible moves after selecting a piece, highlighting of hanging pieces, highlighting of pinned pieces, highlighting of pieces in or entering capture spaces, notification when a player is in check, confirm each move, Auto-Queen when pawn promotes, highlighting last move, sounds, and change theme.	Optional
A pair of users can play a game on a single device.	Desirable
During the game, the user has the option to resign, offer draw, and see previous moves.	Essential
If a player offers a draw, the other player will have the option to accept or decline the draw.	Essential
If the player accepts the draw, then a popup will state “Draw” with the option to play again or go back to the home screen.	Essential
If the player declines the draw, then the other player will be notified that the player declined the draw, and the game will resume.	Essential
After checkmate, resignation, or stalemate, a popup will state “Black wins”, “White wins”, or “Draw” with the option to play again or go back to home screen.	Optional
Home screen will have the option to play a game or go to lessons	Essential
If the user chooses to play a game, then the user will be provided a board to play with a friend locally.	Essential
If the user chooses to go to lessons, then the next recommended lesson will pop up with the remaining lessons underneath.	Desirable
Actor is an unregistered user for account creation.	Essential
Server needs to request email, username, and password.	Essential
A user account with this information is created in the database afterwards.	Essential

If the login is already taken, then the process will loop to prevent the actor from progressing until unique account information is entered.	Essential
If the actor is already signed in, then this specific actor should not be allowed to enter the system and prompt them if they want to sign out instead.	Essential
Actor is a registered user that is not logged in for signing in.	Essential
The actor needs to be prompted with a login page.	Essential
Login page needs to contact the database to validate the user.	Essential
Actor should be redirected to a menu with their account specific features.	Essential
If the actor is signed in, then the specific actor should not be allowed to enter this system and the actor will be prompted if they want to sign out instead.	Essential
If the actor enters invalid information, the interface will prevent the actor from progressing and needs to prompt them about what is invalid (ex: wrong password).	Essential
Actor is a registered user that is logged in for signing out.	Essential
Any other user should not be able to see the option to sign out.	Essential
Server will be contacted that the user wants to sign out.	Essential
Actor is redirected to the view of the page a signed-out user can see.	Essential
Actor is any user (unregistered/registered) for selecting a lesson.	Essential
Actor needs access to a catalog of available lessons.	Essential
Server needs to gather lessons from the database that contains them, handling any connection issues in the process	Essential
Actor can choose to enter a lesson, which hands control to a new display.	Essential
As a lesson progresses, the interface includes different options displayed to the user: a user can flip from one page to the next/previous page, interface accesses pages from the server database, pages need to display text to the user in an easily readable manner, interactive elements (such as small chess boards).	Essential
The interactive elements includes: the user can move pieces in real time, element provides feedback (ex: “good move!”) after an action, element accurately updates the position of pieces, a piece can be “locked” so the user cannot move it.	Desirable
Actor can leave the lesson at any point and save progress if logged in.	Optional

Actor is any user (unregistered/registered) for playing on their own.	Essential
Actor can leave this page at any time.	Desirable
Control alternates between a “white” and a “black” player (white goes first).	Essential
The other player’s pieces are locked during one player’s turn.	Essential
Player is prompted that it is their move.	Desirable
Player can move a piece to a position.	Essential
If the movement is invalid for that piece: do not update the board, prompt the player that the move is invalid, and give the player the ability to move again.	Essential
If the movement is valid for that piece: update the board.	Essential
If the movement captures a piece: the piece is removed from the board.	Essential
Control shifts to the opposing player when the current player makes a valid move.	Essential
If the player is in “check”: a different set of movement rules should be applied so that the user has to address the threat.	Essential
The interface should check if the board state after a move produces either a checkmate or stalemate.	Essential
If there’s a checkmate, the player who last moved is the “winner” of the game.	Essential
If there’s a stalemate, the interface should prompt the players that there is a draw.	Essential
Actor is a registered user for editing user settings.	Essential
All other actors should not have access to the settings menu.	Essential
The actor should be prompted with a list of settings.	Desirable
When the actor updates a setting, the server should receive the request and save the change to the database.	Essential
If a setting is sensitive (“change password”, “update email”): the actor should be prompted to validate their information.	Essential
After three attempts of validating their information, they should be signed out.	Optional
Email confirmation for specific setting changes like password will be sent.	Optional
The user interacts with the system through a mobile or a web browser.	Desirable

Either will execute HTML and Java queries to the web engine using JSP.	Essential
The engine communicates with the database using SQL.	Essential
The database returns the data to the engine which, in turn, returns the resulting package to the browser or application, displaying the results afterwards.	Essential
Individual users have the option to create accounts within the system for tracking personal progress and data.	Essential
Account credentials will need to be stored within the database, made publicly inaccessible, and encrypted to guarantee security.	Essential
No information beyond bare minimum identification and module progress will be stored to minimize the impact of a data leakage should security measures fail.	Essential
System should include accessibility the certain groups of users depend on in order to utilize any software, including but not limited to: text narration, screen readers, variable font size, audio captions, and color correction	Essential

5. Appendixes

None

LearningEDU

ChessEDU
Supplementary Specifications

Version <2.0>

Revision History

Date	Version	Description	Author
25/10/2022	1.0	First Draft	Adair Torres
27/10/2022	2.0	Applied modifications as requested by the Lab Attendant	Adair Torres

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Definitions, Acronyms, and Abbreviations 4
   - 1.3 References 4

2. Supplementary Specifications 5
   - 2.1 Client / Server 5
   - 2.2 Usability 5
     - 2.2.1 Graphical Interface 5
     - 2.2.2 User's Knowledge 5
   - 2.3 Reliability 6
     - 2.3.1 Responsiveness 6
     - 2.3.2 Bug / Defect Tolerance 6
   - 2.4 Performance 6
     - 2.4.1 Execution Speed 6
     - 2.4.2 Resource Use 6
   - 2.5 Supportability 6
     - 2.5.1 Platforms / Operating Systems 6
     - 2.5.2 Maintenance Access 6
   - 2.6 Design Constraints 6
     - 2.6.1 Programming Languages Used 6
   - 2.7 Security 7
     - 2.7.1 Risks 7
     - 2.7.2 User Data Minimization 7
     - 2.7.3 Encryption 7
     - 2.7.4 Credential Requirements 7
   - 2.8 Interfaces 7
     - 2.8.1 Localhost:<port#> 7

Supplementary Specifications

1. Introduction

1.1 Purpose

Supplementary specifications capture the requirements which aren't easily defined within the UseCase Model. Requirements such as: legal standards, quality aspects, reliability, supportability, and execution criteria of the system.

1.2 Definitions, Acronyms, and Abbreviations

Browser

A browser is a software which allows the user to visualize3 and interact with all information presented and flowing through the internet.

Engine

In a software or in a computer, Engine is the term used for smaller programs executing specific functionalities and useful tasks for other programs or software.

Flask

A python module used to establish and control a web application / Engine through HTTP queries and python code. The information and data are controlled by the Engine first and then displayed in its final shape to the end-user through HTML and Jinja templates.

Jinja

A templating engine with special placeholders that allow writing code similar to Python syntax in an HTML web page. A template is passed data to render in the final HTML document.

SQLArchive

A SQLite database specifically formatted to track file data. Entries should include the full file path to a file and data associated with the file, such as access permissions, last modification time, original file size, and any compressed content.

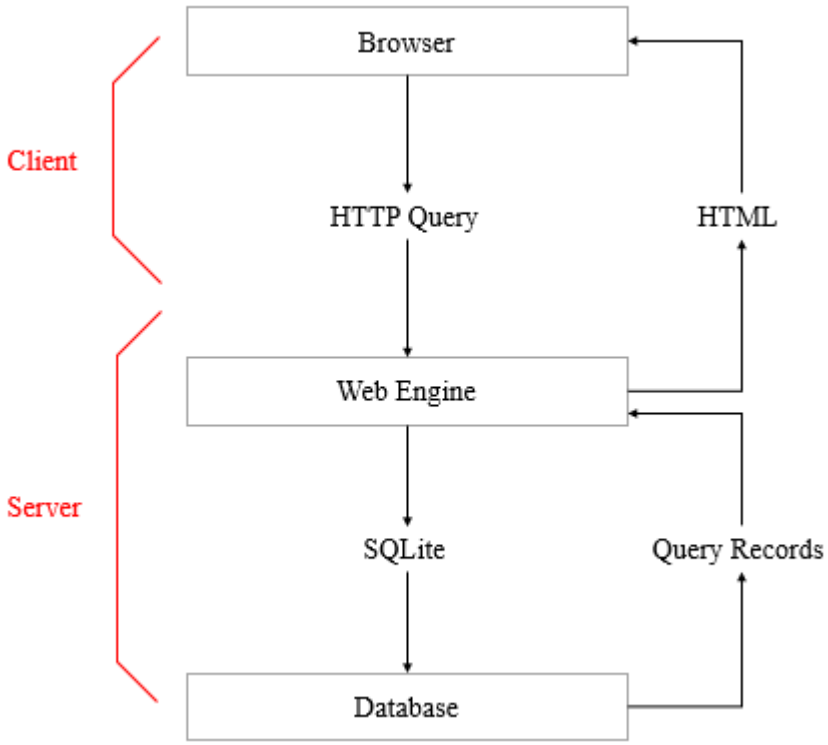
1.3 References

Course Web Page: https://people.eecs.ku.edu/~saiedian/Teaching/448/

2. Supplementary Specifications

2.1 Client / Server

The user interacts with the system through a web browser (Google Chrome, Safari, Mozilla Firefox, etc.). The browser executes HTML and JavaScript queries to the Web Engine through Flask application routing of HTTP methods. The Engine communicates with the Database using SQLite language. Finally the database returns the data to the Engine, which, in turn, gives back the requested package to the browser. The browser then shows the results to the end-user.



2.2 Usability

2.2.1 Graphical Interface

All interactions between a user and the software are made through a graphical interface. Each system functionality must be accessible by mouse or keyboard. The user must be able to choose a specific branch from the system and then obtaining the results on the screen.

2.2.2 User's Knowledge

Targeted users are familiar with the use of web Browsers on the platform / operating system they are utilizing. The tool is designed to be user-friendly, and any user should not require training. Targeted users may have varying levels of understanding and familiarity with the game of chess.

2.2.2.1 Accessibility

The software should include accessibility features for users with disabilities. This includes but is not limited to a narrator to read course modules, announce game piece movements, and board states; a magnifier to increase visibility of web pages; and adaptive controls for users with restricted motor functions.

2.3 Reliability

2.3.1 Responsiveness

The application shall be able to capture, execute, and respond to all user entries. All false entries, such as invalid or restricted HTTP queries, shall be validated and returned to the user without abnormally stopping the application.

2.3.2 Bug / Defect Tolerance

The application must have absolutely no bugs or defects when executing a move in a game of chess. All the rules of Chess must by fully implemented without bugs in order to prevent accidental illegal moves and player frustration. Move and turn logging must also have zero defects in order to maintain the accuracy of game logs.

The application may display some web pages improperly due to certain unexpected screen sizes, aspect ratios, and display formats. However, the application should display consistently across web browser windows, even after a window is resized.

The application should not return the incorrect course module or lesson to the user. The Web Engine should only attempt to locate and track module files through the SQLArchive SQLite database to reduce any chance of a bug resulting from improper file retrieval / storage.

2.4 Performance

2.4.1 Execution Speed

The application will execute itself on a Windows-based platform and response time for all tasks shall be inferior to 1.5 seconds. However, this time may vary depending on the number of users, and request load on the server.

2.4.2 Resource Use

The application will require access to a significant and possibly growing amount of storage memory as more course modules are added by contributors.

2.5 Supportability

2.5.1 Platforms / Operating Systems

Client side can be installed on any platform. While the Server side and Web Engine can be installed on any MacOS, Linux, or Windows system, the current installation targets a Windows-based platform.

2.5.2 Maintenance Access

Maintenance of the software is solely accessible by the LearningEDU developer team. Maintenance of a course / lesson module's content is accessible to Lesson Author users and Lesson Developers.

2.6 Design Constraints

2.6.1 Programming Languages Used

JavaScript and HTML for all web content; Python for Web Engine / Application implementation, SQLite for database management queries.

2.7 Security

2.7.1 Risks

The type of data with the most significant risk are passwords and credentials. As some users reuse passwords, a security breach and leakage of user credentials can lead to accounts not associated with our software becoming compromised.

2.7.2 User Data Minimization

Gather data on individual users should be minimized as much as possible to reduce the liability and risk tied to a data leakage. The bare minimum for any given user is their associated credentials and course module data, including completed and in-progress courses.

2.7.3 Encryption

At minimum, user credentials should be encrypted. This can be either encryption before being placed within the credentials database, or the complete encryption of the credential database.

2.7.4 Credential Requirements

Users should be required to use lengthy and complex passwords for their associated credentials. A user should be required to use a password that includes uppercase and lowercase letters, numbers, and symbols. A password must be at least 12 characters long. At signup, a user should be given tips for creating a strong and memorable password.

2.8 Interfaces

2.8.1 Localhost:<port#>

This interface is used for testing and debugging on developer systems. This is the basic standard for running a Client / Server system on one's own computer and prevents outside interaction so long as the individual device's IP or hostname are not shared. The application is set to use the default port 5000, but can be changed.

LearningEDU

ChessEDU
Use-Case Specifications

Version <2.4>

Revision History

Date	Version	Description	Author
25/10/2022	1.0	First Draft	Adair Torres
26/10/2022	2.1	Reformatting from old requirements document. Most requirements were reworded for clarity or detail. Specified a security requirement for Changing User Settings.	Jack Reynolds
27/10/2022	2.2	Removed blue text.	Jack Reynolds
29/10/2022	2.3	Added Use Case diagrams for Account Creation, Editing User Settings. Specified Use Case Packages.	Jack Reynolds
30/10/2022	2.4	Edited for consistent formatting and spacing, updated table of contents.	Adair Torres

Table of Contents

1. Use-Case Model 6
   - 1.1 Introduction 6
   - 1.2 General Actors Descriptions 6
     - 1.2.1 Unregistered Users 6
     - 1.2.2 Signed-In Users 6
     - 1.2.3 Administrators 6
     - 1.2.4 Product Server Host 6
     - 1.2.5 Credentials Database 6
     - 1.2.6 Lessons Database 6
   - 1.3 Use-Case Model Hierarchy 7
     - 1.3.1 Account Package 7
     - 1.3.2 Interactive Package 7
   - 1.4 Diagrams of the Use-Case Model 8

2. Account Creation 9
   - 2.1 Brief Description 9
   - 2.2 Flow of Events 9
     - 2.2.1 Basic Flow 9
     - 2.2.2 Alternative Flows 9
   - 2.3 Special Requirements 9
   - 2.4 Preconditions 9
     - 2.4.1 Internet Connection 9
     - 2.4.2 Not Signed-In 9
   - 2.5 Postconditions 9
     - 2.5.1 Database Update 9
   - 2.6 Extension Points 10
     - 2.6.1 Automatic Sign-In 10
   - 2.7 Use-Case Diagrams 10
   - 2.8 Other Diagrams 10

3. Signing In 11
   - 3.1 Brief Description 11
   - 3.2 Flow of Events 11
     - 3.2.1 Basic Flow 11
     - 3.2.2 Alternative Flows 11
   - 3.3 Special Requirements 11
   - 3.4 Preconditions 11
     - 3.4.1 Internet Connection 11
     - 3.4.2 Not Signed-In 11
   - 3.5 Postconditions 11
     - 3.5.1 Progress Flags 11
   - 3.6 Extension Points 12
     - 3.6.1 <Optional Register> 12
   - 3.7 Use-Case Diagrams 12
   - 3.8 Other Diagrams 12

4. Signing Out 13
   - 4.1 Brief Description 13

4.2 Flow of Events 13
4.2.1 Basic Flow 13
4.3 Special Requirements 13
4.4 Preconditions 13
4.4.1 Internet Connection 13
4.4.2 Signed-In 13
4.5 Postconditions 13
4.5.1 Progress Flags 13
4.6 Extension Points 13
4.7 Use-Case Diagrams 13
4.8 Other Diagrams 13

5. Selecting a Lesson 14
5.1 Brief Description 14
5.2 Flow of Events 14
5.2.1 Basic Flow 14
5.2.2 Alternative Flows 14
5.3 Special Requirements 14
5.4 Preconditions 14
5.4.1 Internet Connection 14
5.5 Postconditions 14
5.5.1 Progress Flags 14
5.6 Extension Points 15
5.7 Use-Case Diagrams 15
5.8 Other Diagrams 15

6. Playing On Your Own 16
6.1 Brief Description 16
6.2 Flow of Events 16
6.2.1 Basic Flow 16
6.2.2 Alternative Flows 16
6.3 Special Requirements 16
6.4 Preconditions 16
6.4.1 Internet Connection 16
6.5 Postconditions 16
6.6 Extension Points 16
6.7 Use-Case Diagrams 17
6.8 Other Diagrams 17

7. Editing User Settings 18
7.1 Brief Description 18
7.2 Flow of Events 18
7.2.1 Basic Flow 18
7.2.2 Alternative Flows 18
7.3 Special Requirements 18
7.3.1 Security 18
7.4 Preconditions 18
7.4.1 Internet Connection 18
7.5 Postconditions 19
7.5.1 Database Update 19
7.6 Extension Points 19

7.6.1 Automatic Sign Out 19
7.7 Use-Case Diagrams 19
7.8 Other Diagrams 19

Use-Case Specifications

1. Use-Case Model

1.1 Introduction

This system involves the interaction of web client with a web application server that maintains two databases. The list of human actors includes *Users That Are Not Signed-In* or *Unregistered Users* – human actors that are not signed into an account, *Signed-In Users* – human actors that are signed into an account, and *Administrators*. Additionally, there are nonhuman actors, such as the *Product Server Host*, and in some use cases the *Credentials Database* and *Lessons Database* are treated as separate nonhuman actors.

1.2 General Actors Descriptions

1.2.1 Unregistered Users

Unregistered users, **Users not signed in**, and similar phrasing is used for a human actor on a client that is not currently signed in. A distinction may be made between **unregistered** and **not signed in** to differentiate between users that do not have an account in contrast to users that simply are not signed into their account.

These users have limited data associated with them but may have locally stored data indicating progress through lessons.

1.2.2 Signed-In Users

Signed-in users refers to any human actor on a client that is signed in.

These users always have a *username*, *password*, and *email address* associated with them. To allow for the changing of usernames, these users may also have an internally stored *user ID*. Additional information, such as *their name*, is also associated with their account.

1.2.3 Administrators

Administrators refers to any human actor that is associated with the development of ChessEDU.

These users are similar to signed-in users, but they have additional data flags associated with their account that may allow them to bypass restrictions a regular user cannot. They might alternatively access segments of the product from a terminal or separate interface.

1.2.4 Product Server Host

The **Product server host**, most likely referred to as the **server**, **host**, or as **ChessEDU** in a use case, refers to the running instance of the server or the computer this instance of the server is hosted on.

It may communicate with clients that have contacted it through the web, the databases stored on the same device, or may be contacted by an administrator.

1.2.5 Credentials Database

The **Credentials database** refers to the database that stores all information associated with user accounts, such as usernames, passwords, email addresses, and progress on lessons. It can only be contacted through the server.

1.2.6 Lessons Database

The **Lessons database** refers to the database that stores all information associated with lessons, such as HTML pages, scripts, and visual or audio assets stored in separate files. It can only be contacted through

the server.

1.3 Use-Case Model Hierarchy

1.3.1 Account Package

- **Description**

This package contains all functions necessary to the operation of ChessEDU but are not part of the distinguishing functionality of the product.

- **Use Cases**

Account Creation
Signing In
Signing Out
Editing User Settings

- **Actors**

Unregistered Users
Signed-In Users
Administrators
Product Server Host
Credentials Database

- **Relationships**

Total Dependency on Credentials Database – no use cases within this package can function to a satisfactory level without communication between user client and the Credentials Database.

- **Packages Owned**

None.

1.3.2 Interactive Package

- **Description**

This package contains all functions regarding the distinguishing features of the ChessEDU product.

- **Use Cases**

Selecting A Lesson
Playing On Your Own

- **Actors**

Unregistered Users
Signed-In Users
Administrators
Product Server Host
Lessons Database
Credentials Database

- **Relationships**

Total Dependency on Lesson Database – Selecting a Lesson use case cannot function to a satisfactory level without communication between user client and the Lessons Database.
Partial Dependency on Credentials Database – Selecting a Lesson use case lacks desire function (committing progress flags to memory) without communication between user client and the Credentials

Database.

- **Packages Owned**

None.

1.4 Diagrams of the Use-Case Model

None.

2. Account Creation

2.1 Brief Description

ChessEDU as a service can be used without an account, as anyone who can access the website will have access to the free and interactive chess lessons available. For users that want to save their progress through lessons, they can freely register with ChessEDU to log progress in the product's servers.

2.2 Flow of Events

2.2.1 Basic Flow

A user who is not signed-in can create an account by navigating to the "Account" section of the ChessEDU website. Users who are not signed-in will see an option to "Register" in this section of the page. Here they will be prompted to provide an email address, a username, and a password for their account. They will also be prompted to confirm their password. This information will be validated for uniqueness in the credentials database to prevent duplicate accounts, and following unique information, the new account is added to the database. The user is automatically signed-in afterwards.

2.2.2 Alternative Flows

2.2.2.1 User Login Information is Already Taken

In the event that the account information that is provided by the user trying to register – email address or username – is taken by an existing account, the user will be prompted to enter different information for these fields. Both the email address and username need to be available for use. This alternative flow loops until unique information is provided.

2.2.2.2 User is Signed-In

A user who is signed-in will be prompted with a different menu than a user who is not when they navigate to the "Account" section of the ChessEDU webpage. An option to "Sign Out" will be available instead, where the user will be able to see a "Register" menu afterwards.

2.2.2.3 Password and Confirm Password Prompts Do Not Match

A user who provides two password fields that do not match will be alerted that the confirmation of their password failed. The user will then be allowed to enter their password and confirm it again. This process will loop until the confirmation is successful.

2.3 Special Requirements

None.

2.4 Preconditions

2.4.1 Internet Connection

The user must be connected to the internet.

2.4.2 Not Signed-In

The user must not already be signed-in to another account.

2.5 Postconditions

2.5.1 Database Update

The Credentials database is always updated after a successful execution to include a new instance of

Account.

2.6 Extension Points

2.6.1 Automatic Sign-In

Following successful execution of this use case, the behavior of the “Signing In” use case is extended into Account Creation.

2.7 Use-Case Diagrams

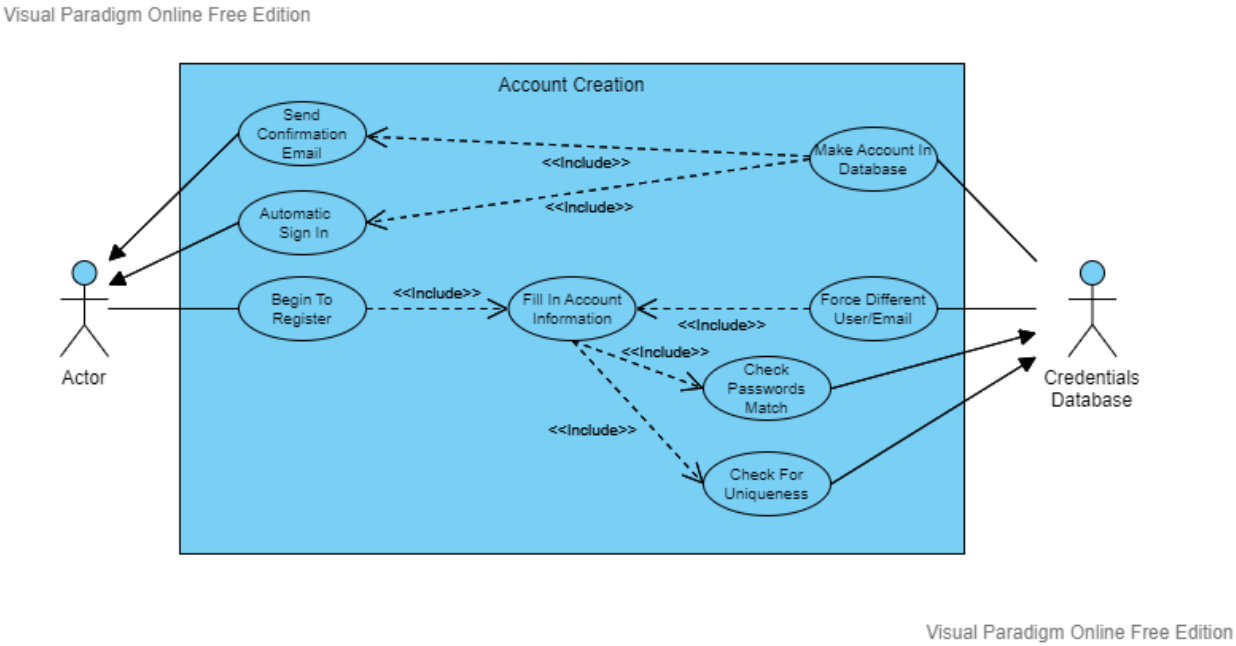


Figure 1 Account Creation.

2.8 Other Diagrams

None.

3. Signing In

3.1 Brief Description

A user that has created an account with ChessEDU can utilize the unique features of having an account by signing in. This process also verifies that the account created by the user remains in control of the user, forcing them to prove ownership of the account on sign in.

3.2 Flow of Events

3.2.1 Basic Flow

A user that is not already signed-in can do so by selecting the "Account" section on the ChessEDU website. Under this section, this user will be able to see an option to "Sign In" to an existing account. Selecting this option will redirect the user to a separate page that requires the user to input their username and password. This information is relayed to the Credentials database, which will verify that the username exists under some account and the password entered matches the password associated with this account. If the username can be found and the password is correct, the user is signed-in and now has access to menus and other information exclusive to users with accounts.

3.2.2 Alternative Flows

3.2.2.1 Username for Account Cannot be Found

If the user enters an invalid username – a username that is not associated with any account in the Credentials database – the user will be alerted that the account they are trying to access does not exist. They will also receive a message that they can alternatively register for a new account. The user can attempt to sign in again.

3.2.2.2 Password Entered Does Not Match Stored Password

If the user enters an invalid password – a password that does not match the password associated with the account that the username is – the user will be alerted that the password or username is incorrect. The user can attempt to sign in again.

3.3 Special Requirements

None.

3.4 Preconditions

3.4.1 Internet Connection

The user must be connected to the internet.

3.4.2 Not Signed-In

The user must not already be signed-in to another account. Any user already signed in will not be able to see the option to sign in.

3.5 Postconditions

3.5.1 Progress Flags

Following sign in, any lessons the user accesses while signed-in will log the progress made by the user. Any progress flags set by the user after completing a section of a lesson will be committed to the Credentials database.

3.6 Extension Points

3.6.1 Optional Register

The user interface will prompt a user attempting to sign into an account that does not exist with the option to register instead. This will redirect the user to the “Register” use case.

3.7 Use-Case Diagrams

None.

3.8 Other Diagrams

None.

4. Signing Out

4.1 Brief Description

A user with a ChessEDU account, to protect the security of their account, may want to sign out from a device. This option also provides users with the ability to switch to a different account, especially helpful if more than one user uses the same device.

4.2 Flow of Events

4.2.1 Basic Flow

A user who is signed-in can access the option to sign out from the "Account" section of the ChessEDU website. Under this section, a user who is signed-in will be able to see the option to "Sign Out." Selecting this option will update the page and contact the server that the user is no longer signed in from this device. Following confirmation from the server, the user will now see the same options as any user without an account.

4.3 Special Requirements

None.

4.4 Preconditions

4.4.1 Internet Connection

The user must be connected to the internet.

4.4.2 Signed-In

The user must be signed in to see this option. Any user not signed in will have no option to "Sign Out" displayed.

4.5 Postconditions

4.5.1 Progress Flags

Following sign out, any progress flags the user would send to the Credentials database are not committed to any account. If local data is deleted or the user signs into their account on another device, this progress will be lost.

4.6 Extension Points

None.

4.7 Use-Case Diagrams

None.

4.8 Other Diagrams

None.

5. Selecting a Lesson

5.1 Brief Description

The main service that ChessEDU provides is free, interactive lessons on the ChessEDU website. These lessons can be accessed by anyone online, and contain written explanations as well as interactive elements, such as chess board simulations. Individual lessons can be accessed from a main menu and will save user progress upon completion.

5.2 Flow of Events

5.2.1 Basic Flow

A user (signed-in or not) can access lessons by navigating to the "Lessons" section on the ChessEDU website. This will direct the user to a page that displays the list of available lessons that contain a lesson name and a short description of the lesson. To build this page, the client instance sends a request for basic data on lessons to the Lessons database, which returns the summary information for each lesson in the database. A user can then select an available lesson, which will send a request for the total lesson information for that specific lesson to the Lessons database. The client will build the lesson from these lesson files.

Within a given lesson, a user can interact with several different elements of the page. Depending upon the lesson, the user may have access to the option to navigate forward or backward through pages and manipulate an interactive chess board. The page needs to be able to display these elements in an easy-to-read format, and update with requests from the user.

The interactive chess board element, depending on the lesson, may be of varying sizes or features. Pieces must be movable, and upon making a move the board may provide several different kinds of feedback ("Good move!", "That was a mistake!", etc.). Some pieces may be "locked" by the interactive element as pieces the user cannot interact with on that move, such as if the piece belongs to the opponent or if the lesson wants the user to learn how to use a specific piece or tactic.

If the user completes a section of a lesson, this information will be saved locally so the user may continue to the next lesson. A user may back out at any time to return to the list of available lessons.

5.2.2 Alternative Flows

5.2.2.1 User is Signed-In

A user that is signed-in will have data that marks their progression through lessons (progress flags) stored in server data rather than locally. This data is committed to the Credentials database and updates the record of progress stored in the user's account.

5.3 Special Requirements

None.

5.4 Preconditions

5.4.1 Internet Connection

The user must be connected to the internet.

5.5 Postconditions

5.5.1 Progress Flags

Upon completion of a lesson or a section of a lesson, new data is created locally that saves the progress of

the user. This data, depending on the user, may also be saved to the ChessEDU servers.

5.6 Extension Points

None.

5.7 Use-Case Diagrams

None.

5.8 Other Diagrams

None.

6. Playing On Your Own

6.1 Brief Description

The other major service that ChessEDU as a product provides is local play – the option to practice a game against another person in the same area. The ChessEDU website will provide a full chessboard as an option for all users, which can be used to run a game in a standard format on the user's device.

6.2 Flow of Events

6.2.1 Basic Flow

A user (signed-in or not) can access the ability to play a game by navigating to the "Practice" section of the ChessEDU website. Here the user will be provided with a full interactive chessboard, as well as any options that they may want to configure before starting a game.

Control flow alternates between a "white" player and a "black" player during the game, but both are controlled by the same device. During one player's turn, the other player's pieces are "locked" and cannot be manipulated by the user. Once the user makes a valid move for the piece they select, the board is updated, including any possible captures.

The page also will regularly check for special board states, such as Check, Checkmate, and Stalemate. If Checkmate or Stalemate is found, the user is notified that the game is over and of the results of the game.

The user may also exit the page at any time.

6.2.2 Alternative Flows

6.2.2.1 User Makes an Invalid Move

If the user makes a move that is disallowed for some reason – the piece does not belong to the player that currently has their turn, the movement is impossible for that piece, the user makes a move that would be allowed under normal circumstances but is not allowed while the current player's King is in check – then the page alerts the user that this move is not allowed, and gives them the ability to move again. This alternative flow loops until a valid move is made by the user.

6.3 Special Requirements

None.

6.4 Preconditions

6.4.1 Internet Connection

The user must be connected to the internet.

6.5 Postconditions

None.

6.6 Extension Points

None.

6.7 Use-Case Diagrams

None.

6.8 Other Diagrams

None.

7. Editing User Settings

7.1 Brief Description

Alongside storing lesson progress with accounts, ChessEDU accounts can be updated with new information as needed. Users who may want to edit their account (such as to change their password or email) can update their information online within the product.

7.2 Flow of Events

7.2.1 Basic Flow

A user who is signed-in can access the option to change their settings from the "Account" section on the ChessEDU website. On the Account page or a connected page, the user will be able to see their settings associated with their account and will be shown options next to any configurable setting to update that setting.

If a setting is not sensitive – entailing the user's security or ability to access their account – the user will have the ability to edit the option using some web input (buttons, text box, etc.) on the same page. The user will then confirm the change, which will cause their client to send a request to the Credentials database to update this information on their profile. A confirmation will be sent back to the user.

If a setting would involve the security of the user's account – such as the user's account's associated username, password, or email address – the user will be redirected to a separate page to verify themselves. Here they will be prompted to enter their password, or alternatively access a page through their email if they wish to reset their password. Successful authentication will similarly update these features.

7.2.2 Alternative Flows

7.2.2.1 User Fails to Authenticate Password

If the user fails to verify their password when changing a sensitive setting, the user will be alerted that the have entered the wrong password for their account. They will be allowed to try again after this. After a set number of failures, the user will be automatically signed out on their device.

7.3 Special Requirements

7.3.1 Security

The user should not have a means to edit any setting that would change how they sign in without some form of security barrier, either through their password or their email address. The security barrier should be easy for an average user to succeed – such as entering their current password or accessing a page through their email address – but be difficult for a user that did not create the account.

The associated password or email address should not be able to be easily updated with just the current password or the current email address respectively. A user should need to know the opposing piece of information to edit the setting, so that a user that maliciously gains the current password but not the current email address cannot gain control of the account, and vice versa. A possible security barrier could include a time restriction on how quickly the user can edit both of these settings.

7.4 Preconditions

7.4.1 Internet Connection

The user must be connected to the internet.

7.5 Postconditions

7.5.1 Database Update

Upon a setting change, the Credentials database will be updated with new information, possibly affecting how the user would sign in next.

7.6 Extension Points

7.6.1 Automatic Sign Out

If a user fails to verify their security information, the functionality of the “Sign Out” use case may be extended to this use case.

7.7 Use-Case Diagrams

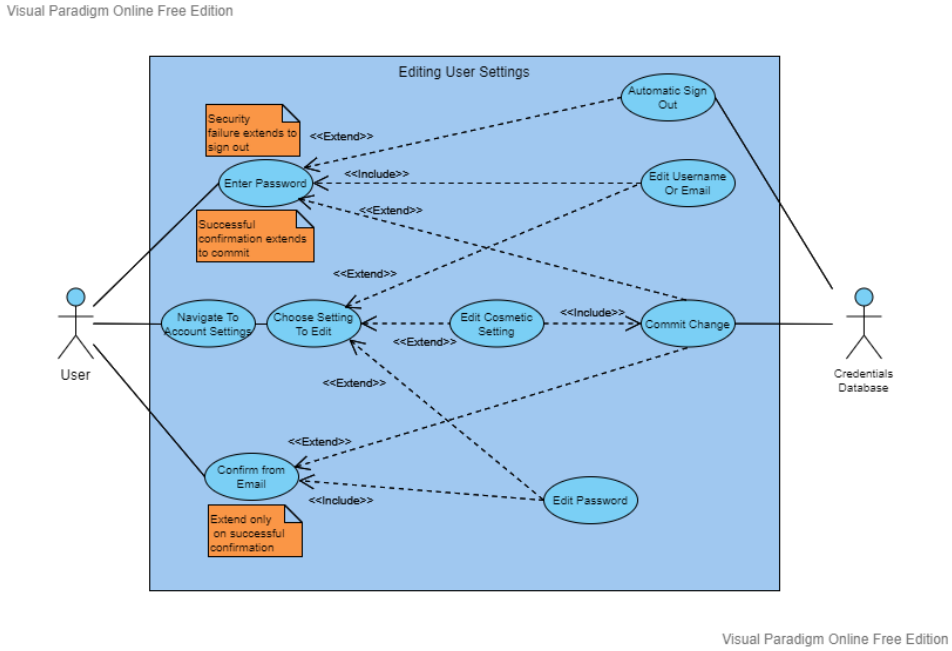


Figure 2 Editing User Settings.

7.8 Other Diagrams

None.

LearningEDU

ChessEDU
Software Architecture Document

Version <2.1>

Revision History

Date	Version	Description	Author
25/10/2022	1.0	First Draft	Adair Torres
29/10/2022	1.1	Reformatted old document to new document.	Chinh Nguyen
30/10/2022	2.1	Improved Class Diagrams and fleshed out Architecture descriptions.	Adair Torres

Table of Contents

1. Introduction 4
   1.1 Purpose 4
   1.2 Scope 4
   1.3 Definitions, Acronyms, and Abbreviations 4
   1.4 References 4
   1.5 Overview 4

2. Architectural Representation 4

3. Architectural Goals and Constraints 4

4. Use-Case View 5
   4.1 Use-Case Realization 5

5. Logical View 5
   5.1 Overview 5
   5.2 Architecturally Significant Design Packages 6
      5.2.1 Design Model: Design Class Diagrams 6
      5.2.2 Design Classes Description 7

6. Interface Description 14

7. Size and Performance 14

8. Quality 14

Software Architecture Document

1. Introduction

1.1 Purpose

The purpose of this document is to detail the architecture design of the ChessEDU application. It serves as a means establishing the overlying architecture of ChessEDU and the design decisions made. This document uses various architectural views to display the different components of the software product.

1.2 Scope

This Software Architecture Document offers an architectural summary of the ChessEDU product. ChessEDU is a web browser based chess learning and development service. ChessEDU allows users to track course and module progress, as well as practice chess maneuvers while learning or against a local opponent.

1.3 Definitions, Acronyms, and Abbreviations

See Glossary, document chessedu_gloss..pdf

1.4 References

1. ChessEDU – Glossary
2. ChessEDU – Use-Case Specifications
3. ChessEDU – Supplementary Specifications
4. ChessEDU – Software Requirements Specifications

1.5 Overview

This document contains information regarding the general architecture of ChessEDU and overlying details of the project's organizational structure.

2. Architectural Representation

This document presents the architecture as a series of use-case view and object class diagrams. These diagrams use the Unified Modeling Language (UML).

3. Architectural Goals and Constraints

The ChessEDU application is a stand-alone web service that is accessible through a user's web browser. Its major components consist of: a web Engine, credential and archive databases, and a course file system.

All components must execute on a developer personal computer for testing purposes and function and a production server(s) for deployment.

Server and Database components can exist on separate hosts or a singular host device, depending on memory storage requirements and efficiency.

The web Engine and supplied course web pages must function on various types of browsers, including but not limited to Google Chrome, Mozilla Firefox, and Safari.

4. Use-Case View

The Use-Case View is a set of scenarios and/or use cases that are considered vital information in analyzing the process and functionality of an iteration. It describes the set of scenarios and/or use cases that represent some significant, core functionality. This also include use cases that

Refer to *Use-Case Specifications* document for more information – chessedu_ucspec..pdf

4.1 Use-Case Realization

To be implemented in a Use-Case Realization document later on.

5. Logical View

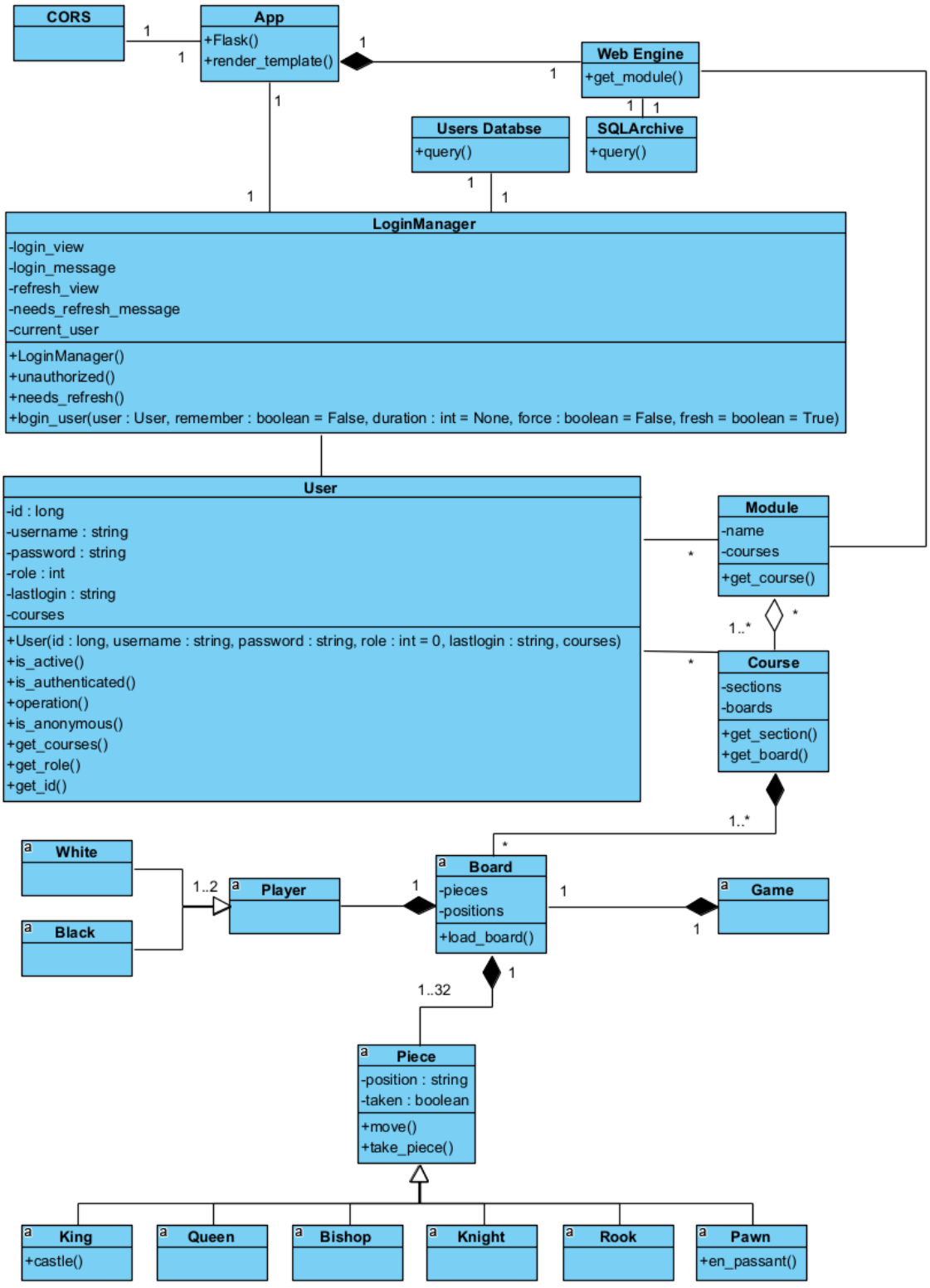
This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities.

5.1 Overview

This subsection describes the overall decomposition of the design model in terms of its package hierarchy and layers.

5.2 Architecturally Significant Design Packages

5.2.1 Design Model: Design Class Diagrams



5.2.2 Design Classes Description

Property	Description
Name	CORS
Description	A class required by Flask for testing on developer personal computers. To be removed in production.
Responsibilities	None that need detailed information
Relations	Connects to a single Flask application.
Methods	None
Attributes	None
Special Requirements	None

Property	Description
Name	App
Description	A class representing an initialized Flask application.
Responsibilities	Manages URL routing and generation for the main pages of the web interface.
Relations	Associated with a LoginManager and composed of a Web Engine
Methods	render_template(): Loads a template passed as a parameter.
Attributes	None
Special Requirements	Must be passed to a CORS object for local device testing.

Property	Description
Name	Web Engine
Description	Object that handles the retrieval of web documents.
Responsibilities	Receives input from the Flask object and retrieves requesting documentation from file system.
Relations	Associated to an SQLArchive object.
Methods	get_module(): Retrieves a module from the SQLArchive and returns it to the Flask object.
Attributes	None
Special Requirements	None

Property	Description
Name	SQLArchive
Description	An SQLArchive object that handles queries for the SQLArchive database system.
Responsibilities	Interacts with the SQLArchive database to retrieve the full pathname for a target file.
Relations	Associated to a Web Engine that requests file paths.
Methods	query(): Takes a passed filename and queries the database for the full path to the file.
Attributes	None
Special Requirements	None

Property	Description
Name	LoginManager
Description	A Flask object that handles logging in a user.
Responsibilities	Manages the refresh timer for a user's session.
Relations	Associated to a single App object, User objects, and a Users Database object.
Methods	unauthorized(): Redirects a user attempting to access and unauthorized view. needs_refresh(): Set how long a user's session remains active before going stale. login_user(): Logs a user in.
Attributes	login_view: The view a user is directed to after logging in. login_message: A message displayed to the user upon login. refresh_view: The amount of time before a view requires refresh. needs_refresh_message: A message displayed to the user when their session requires a refresh. current_user: The current user managed by the LoginManager.
Special Requirements	None

Property	Description
Name	Users Database
Description	A Users Database object that handles queries for the user credentials database system.
Responsibilities	Interacts with the user-credentials database to retrieve the details for a user who successfully logs in..
Relations	Associated to a LoginManager that logs users in.
Methods	query(): Takes a passed username and password and queries the database for the matching entry to verify against.
Attributes	None
Special Requirements	None

Property	Description
Name	User
Description	An object representing a User accessing the web interface.
Responsibilities	Tracks attributes assigned to a user.
Relations	Associated to a LoginManger, a User is associated to their history of Modules and Courses.
Methods	is_active(): Required function by flask-login is_anonymous(): Required function by flask-login, should always return False. is_authenticated(): Required function by flask-login, used in @login_required Flask routes. is_active(): Required function by flask-login, all users should be active until their session needs a refresh. get_courses(): Returns a list of the courses associated to the User. get_role(): Returns the user's role attribute. get_id(): Returns the user's id attribute.
Attributes	Id: A unique id assigned to the user used in database queries. username: A username chosen by the user that they are referred to as. password: A 12-15 private character string used to login a user. role: An integer assigned to the user used to determine their access rights. lastlogin: A string date format that tracks when the user last logged in. courses: a list of courses the user has taken.
Special Requirements	None.

Property	Description
Name	Module
Description	An object representing a set of courses grouped together based on a shared topic.
Responsibilities	None that need detailed information.
Relations	A Module is an aggregation of one or more Courses, associated to a User, and is associated to a Web Engine that retrieves them.
Methods	get_course(): Retrieves a specific Course within a Module.
Attributes	name: A unique name given to a Module that summarizes its focus. courses: A list of the Courses that make up a Module.
Special Requirements	None

Property	Description
Name	Course
Description	An object representing a Course available for a User to take.
Responsibilities	None that need detailed information.
Relations	An aggregation of Courses comprises a Module, is associated to a User, and can be composed of zero or more Boards.
Methods	get_section(): Returns a section of text from the course to be displayed on an html page. get_board(): Returns a pregenerated board for a User to interact with.
Attributes	sections: A list that organizes chunks of information or text within a Course. boards: A list of the boards a Course displays to the User to interact with.
Special Requirements	None.

Property	Description
Name	Board
Description	An object that represents a board state in a game of chess.
Responsibilities	Tracks the positions of pieces across the board.
Relations	Boards may be part of a Course's composition compose a Game, and are composed of 1 or 2 players and between 1 and 32 pieces.
Methods	load_board(): Returns the data in a board object to load a chessboard on a HTML page through JavaScript.
Attributes	pieces: A list of the pieces that initially spawn on the chessboard. positions: A list of the positions of each individual piece that initially spawn on the chessboard.
Special Requirements	The pieces and position attribute lists must be of equal length, as a piece and its position share an index.

Property	Description
Name	Game
Description	An abstract object used to represent a full Game of chess.
Responsibilities	None that need detailed information.
Relations	A Game is composed of a single board.
Methods	None
Attributes	None
Special Requirements	A Game object is used when a User practices a new game of chess, and is typically only created by the web interface when the User wants to play a full game.

Property	Description
Name	Player
Description	An abstract class used to differentiate between multiple players in a game of chess.
Responsibilities	Establishes the User as either the White or Black Player on a board, another Player may take the leftover role.
Relations	A Player is an abstract player representing either the White or Black side of the board.
Methods	None
Attributes	None
Special Requirements	None

Property	Description
Name	White
Description	Abstract class representing the White side of a Board.
Responsibilities	None that need detailed information.
Relations	A generalization of a Player.
Methods	None
Attributes	None
Special Requirements	The White side of a board always has the first move.

Property	Description
Name	Black
Description	Abstract class representing the Black side of a Board.
Responsibilities	None that need detailed information.
Relations	A generalization of a Player.
Methods	None
Attributes	None
Special Requirements	The Black side of a board always has the second move.

Property	Description
Name	Piece
Description	An abstract class representing a Piece on a Board in a Game of chess.
Responsibilities	Tracks a piece's position on a board and whether the piece has been taken or removed from the board.
Relations	A Board is composed of between 1 and 32 pieces. A Piece is a generalization of its six possible types.
Methods	move(): Defined by how each type of piece can move. take_piece(): Called when a Piece moves to a space occupied by another Piece in order to remove it from the board.
Attributes	position: A two character string that describes a Piece's position on the Board. taken: A Boolean value that tells whether a piece has been taken.
Special Requirements	The move() function must be defined by one of the Piece subclasses.

Property	Description
Name	King
Description	A class representing a King piece in a game of chess.
Responsibilities	None that need detailed information.
Relations	An implementation of a Piece as a King.
Methods	move(): Implemented such that a King can only move one space in a given direction. castle(): A special maneuver for Kings.
Attributes	None
Special Requirements	A Player is forced to move a King out of check during their turn, and loses the game if their King is put into checkmate.

Property	Description
Name	Queen
Description	A class representing a Queen piece in a game of chess.
Responsibilities	None that need detailed information.
Relations	An implementation of a Piece as a Queen.
Methods	move(): Implemented such that a Queen can move any number of spaces in a given direction.
Attributes	None
Special Requirements	None

Property	Description
Name	Bishop
Description	A class representing a Bishop piece in a game of chess.
Responsibilities	None that need detailed information.
Relations	An implementation of a Piece as a Bishop.
Methods	move(): Implemented such that a Bishop can move any number of spaces in a diagonal direction.
Attributes	None
Special Requirements	None

Property	Description
Name	Knight
Description	A class representing a Knight piece in a game of chess.
Responsibilities	None that need detailed information.
Relations	An implementation of a Piece as a Knight.
Methods	move(): Implemented such that a Knight can move in an L shaped pattern.
Attributes	None
Special Requirements	None

Property	Description
Name	Rook
Description	A class representing a Rook piece in a game of chess.
Responsibilities	None that need detailed information.
Relations	An implementation of a Piece as a Rook.
Methods	move(): Implemented such that a Rook can move any number of spaces in a horizontal or vertical direction.
Attributes	None
Special Requirements	Moves in a special way when a King makes a castling maneuver.

Property	Description
Name	Pawn
Description	A class representing a Pawn piece in a game of chess.
Responsibilities	None that need detailed information.
Relations	An implementation of a Piece as a Pawn.
Methods	move(): Implemented such that a Pawn can move a single space forward or two spaces forward from its starting row. en_passant(): A special maneuver for pawns.
Attributes	None
Special Requirements	A pawn can become another piece upon reaching the opposite side of the board.

6. Interface Description

To be implemented in a User Interface document later on.

7. Size and Performance

The chosen architecture supports the sizing and timing requirements through the implementation of a client-server architecture. The client portion is handled through a User's web browser that interacts with the Web Engine to retrieve HTML pages for Modules and Courses. The Server portion is managed by two databases that cooperate with various classes associated with a User to log users in and display their desired content along with generated boards. The components have been designed to target minimal disk and memory requirements necessary for a user's personal computer.

8. Quality

The software architecture supports the quality requirements, as mentioned in the Software Requirements Specification and Supplementary Specification.

LearningEDU

ChessEDU
Use-Case-Realization Specifications

Version <1.3>

Revision History

Date	Version	Description	Author
31/10/22	1.0	First Draft	Grant Jones
2/11/22	1.1	Added Initial Use Cases.	Adair Torres
3/11/22	1.2	Added Use-Case Sequence Diagrams	Adair Torres
4/11/22	1.3	Added details to Use-Case Realizations	Adair Torres

Table of Contents

1. Introduction 5
   - 1.1 Purpose 5
   - 1.2 Scope 5
   - 1.3 Definitions, Acronyms, and Abbreviations 5
   - 1.4 References 5
   - 1.5 Overview 5

2. USE CASE <Sign Up> 6
   - 2.1 Flow of Events - Design 6
   - 2.2 Interaction Diagrams 6
     - 2.2.1 Sequence Diagrams 6
     - 2.2.2 Collaboration Diagrams 7
     - 2.2.3 Participating objects 7
   - 2.3 Class Diagrams 7
   - 2.4 Derived Requirements 7

3. USE CASE <LOGIN> 8
   - 3.1 Flow of Events - Design 8
   - 3.2 Interaction Diagrams 8
     - 3.2.1 Sequence Diagrams 8
     - 3.2.2 Collaboration Diagrams 8
     - 3.2.3 Participating objects 8
   - 3.3 Class Diagrams 9
   - 3.4 Derived Requirements 9

4. USE CASE <LOGOUT> 10
   - 4.1 Flow of Events - Design 10
   - 4.2 Interaction Diagrams 10
     - 4.2.1 Sequence Diagrams 10
     - 4.2.2 Collaboration Diagrams 10
     - 4.2.3 Participating objects 10
   - 4.3 Class Diagrams 11
   - 4.4 Derived Requirements 11

5. USE CASE <Load Course> 12
   - 5.1 Flow of Events - Design 12
   - 5.2 Interaction Diagrams 12
     - 5.2.1 Sequence Diagrams 12
     - 5.2.2 Collaboration Diagrams 13
     - 5.2.3 Participating objects 13
   - 5.3 Class Diagrams 14
   - 5.4 Derived Requirements 14

6. USE CASE <Local Game> 15
   - 6.1 Flow of Events - Design 15
   - 6.2 Interaction Diagrams 15
     - 6.2.1 Sequence Diagrams 15
     - 6.2.2 Collaboration Diagrams 16

6.2.3 Participating objects 16
6.3 Class Diagrams 17
6.4 Derived Requirements 17

Use-Case-Realization Specification

1. Introduction

1.1 Purpose

This document grants a detailed overview of the system using several diagrams representing the system functions.

1.2 Scope

ChessEDU will allow a user to learn and improve at the game of chess at their own pace by providing an interface to view courses and modules. A user's progress through a module will be tracked and saved in order for a user to resume a lesson from where they last left off. This Use-Case Realization document provides an overview of the use cases developed in ChessEDU.

1.3 Definitions, Acronyms, and Abbreviations

See Glossary, document chessedu_gloss.pdf

1.4 References

1. ChessEDU – Glossary
2. ChessEDU – Use-Case Specifications
3. ChessEDU – Supplementary Specifications

1.5 Overview

The sections of the Use-Case Realization document describes use-cases in terms of their flow of events, participant objects, and corresponding diagrams.

2. USE CASE <Sign Up>

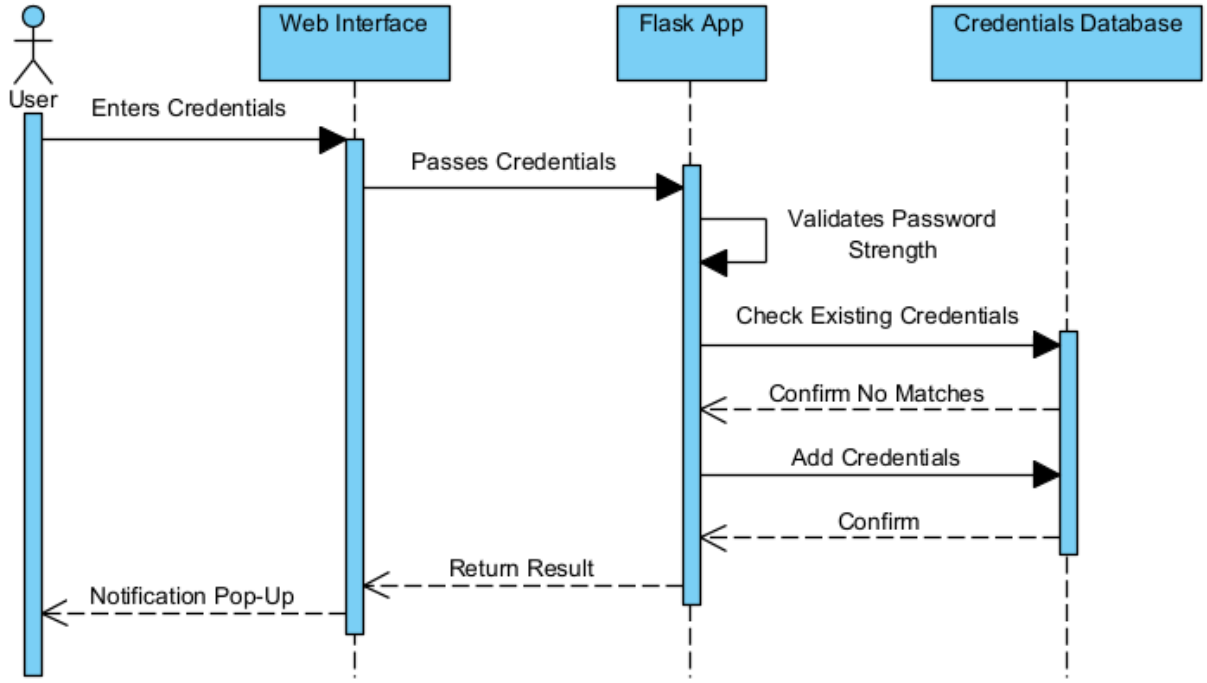
2.1 Flow of Events - Design

Upon navigating to the ChessEDU web application's login page, the user is given a page where they can enter username and password credentials to create an account for the service. The strength of the password is first validated, requiring the user to select a password at least 8 characters in length and containing uppercase and lowercase letters, numbers, and symbols. Afterwards, the credentials database is checked to ensure that no account with the same credentials already exists and returns the result.

2.2 Interaction Diagrams

2.2.1 Sequence Diagrams

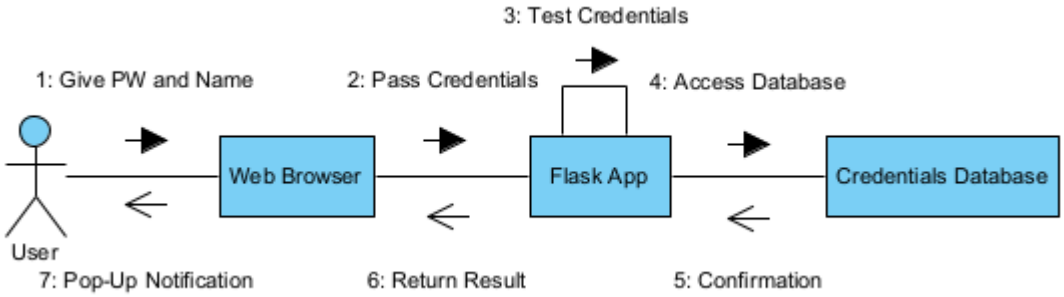
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Sign Up>.



[Figure 1: Sequence Diagram: User Sign Up]

2.2.2 Collaboration Diagrams

This Collaboration Diagram shows the static structure of the Use-Case <User Sign Up>.



[Figure 2: Collaboration Diagram: Sign Up

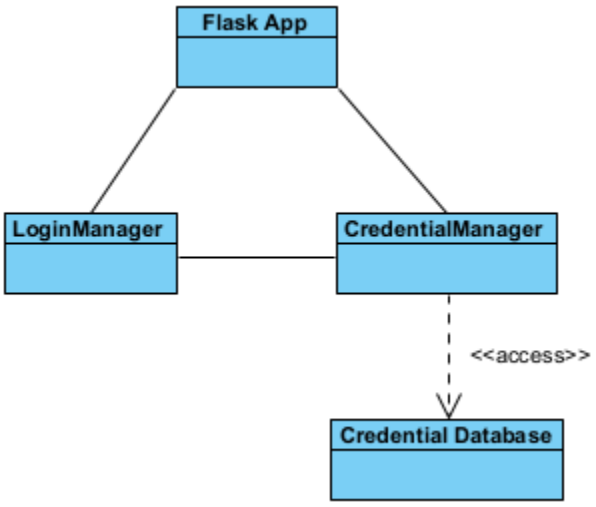
2.2.3 Participating objects

The following objects collaborate and define the Use-Case <Sign Up>:

Web Browser	This object represents the tool used to access the system and the visible part of the application.
Flask App	This object handles HTTP methods and requests and acts as the system's REST API to load web browser pages and redirect the user.
Credentials Database	This object receives SQLite queries to retrieve user credentials and information.

2.3 Class Diagrams

The following Class Diagram shows the relations and constraints between Classes and Objects involved in the Use-Case.



[Figure 3: Class Diagram: Sign Up]

2.4 Derived Requirements

- Attempts to create an account with an existing username or password must be rejected.

3. USE CASE <LOGIN>

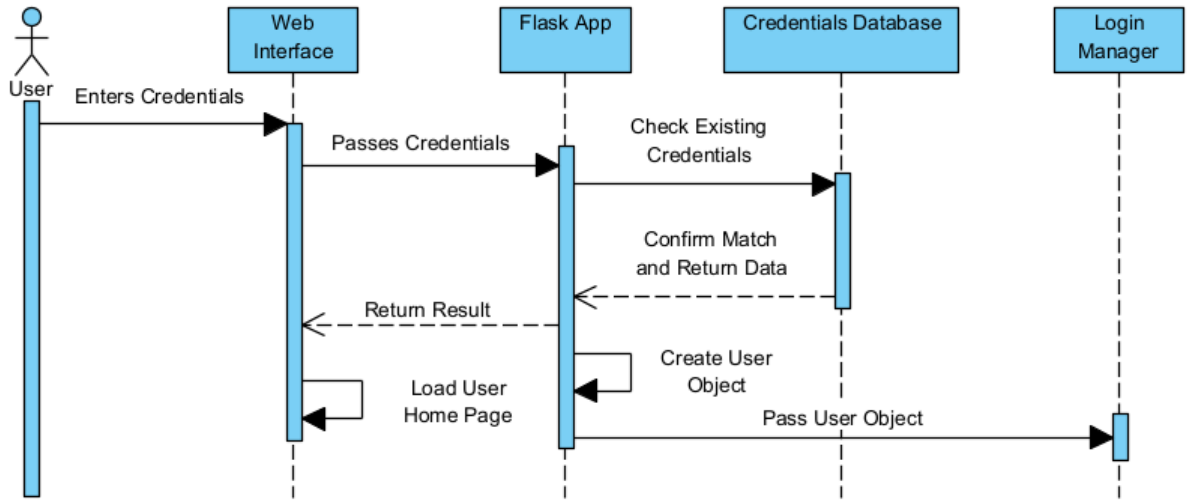
3.1 Flow of Events - Design

A user who has previously created an account navigates to the login page. The user then enters and submits their credentials. If successfully validated against the credentials database, a User object is authenticated and logged in to the LoginManager, redirecting the user to their home page afterwards.

3.2 Interaction Diagrams

3.2.1 Sequence Diagrams

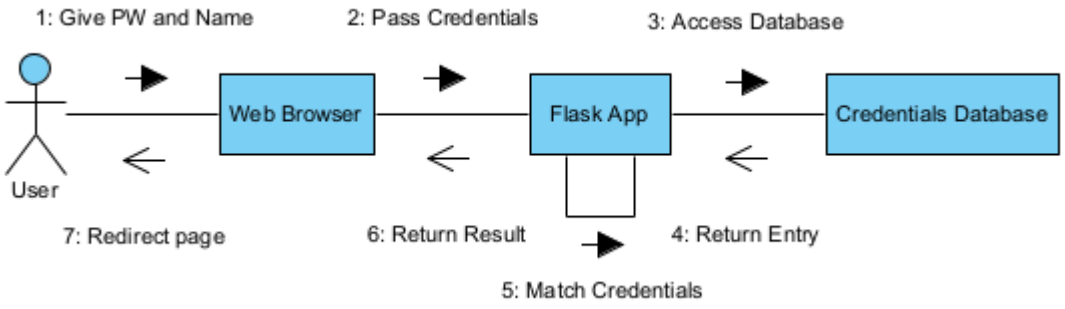
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Login>.



[Figure 4: Sequence Diagram: Login]

3.2.2 Collaboration Diagrams

This Collaboration Diagram shows the static structure of the Use-Case <Login>.



[Figure 5: Collaboration Diagram: Login]

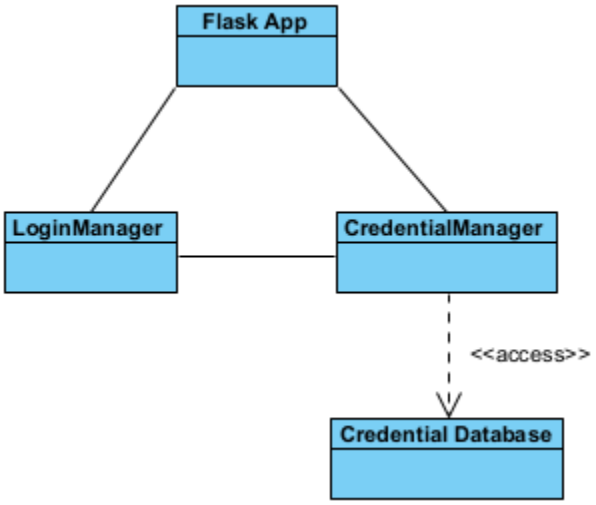
3.2.3 Participating objects

The following objects collaborate and define the Use-Case <Login>:

Web Browser	This object represents the tool used to access the system and the visible part of the application.
Flask App	This object handles HTTP methods and requests and acts as the system's REST API to load web browser pages and redirect the user.
Credentials Database	This object receives SQLite queries to retrieve user credentials and information.

3.3 Class Diagrams

The following Class Diagram shows the relations and constraints between Classes and Objects involved in the Use-Case.



[Figure 6: Class Diagram: Login]

3.4 Derived Requirements

- An authenticate / logged in user must automatically be redirected to their home page upon visiting the login page.

4. USE CASE <LOGOUT>

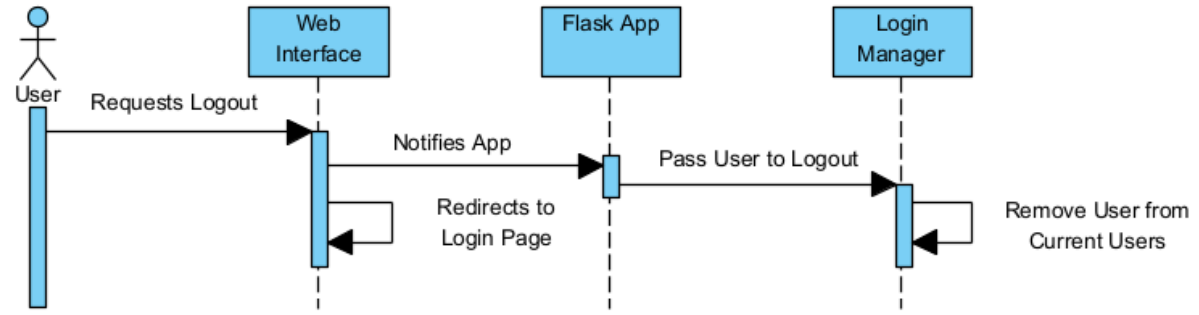
4.1 Flow of Events - Design

An authenticated user has finished using the system and manually wants to logout of the system.

4.2 Interaction Diagrams

4.2.1 Sequence Diagrams

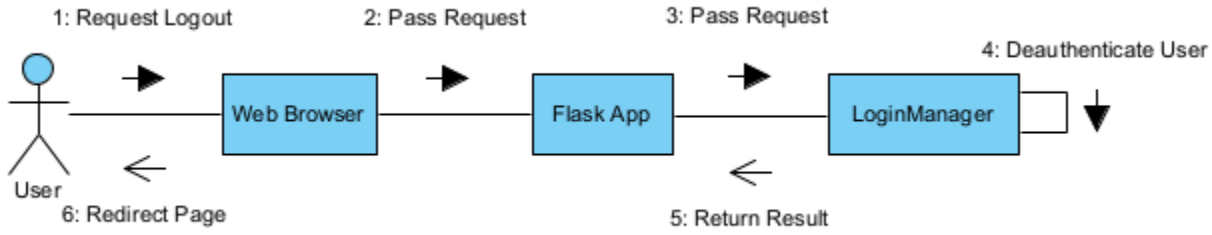
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Logout>.



[Figure 7: Sequence Diagram: Logout]

4.2.2 Collaboration Diagrams

This Collaboration Diagram shows the static structure of the Use-Case <Logout>.



[Figure 8: Collaboration Diagram: Logout]

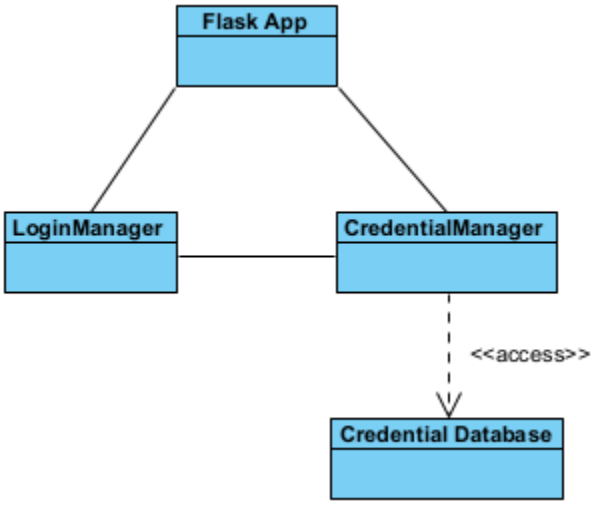
4.2.3 Participating objects

The following objects collaborate and define the Use-Case <Logout>:

Web Browser	This object represents the tool used to access the system and the visible part of the application.
Flask App	This object handles HTTP methods and requests and acts as the system's REST API to load web browser pages and redirect the user.
LoginManager	This object tracks authenticated users and removes them once they have logged out.

4.3 Class Diagrams

The following Class Diagram shows the relations and constraints between Classes and Objects involved in the Use-Case.



[Figure 9: Class Diagram: Logout]

4.4 Derived Requirements

- Any new course progress or account setting changes must be saved as the user logs out.

5. USE CASE <Load Course>

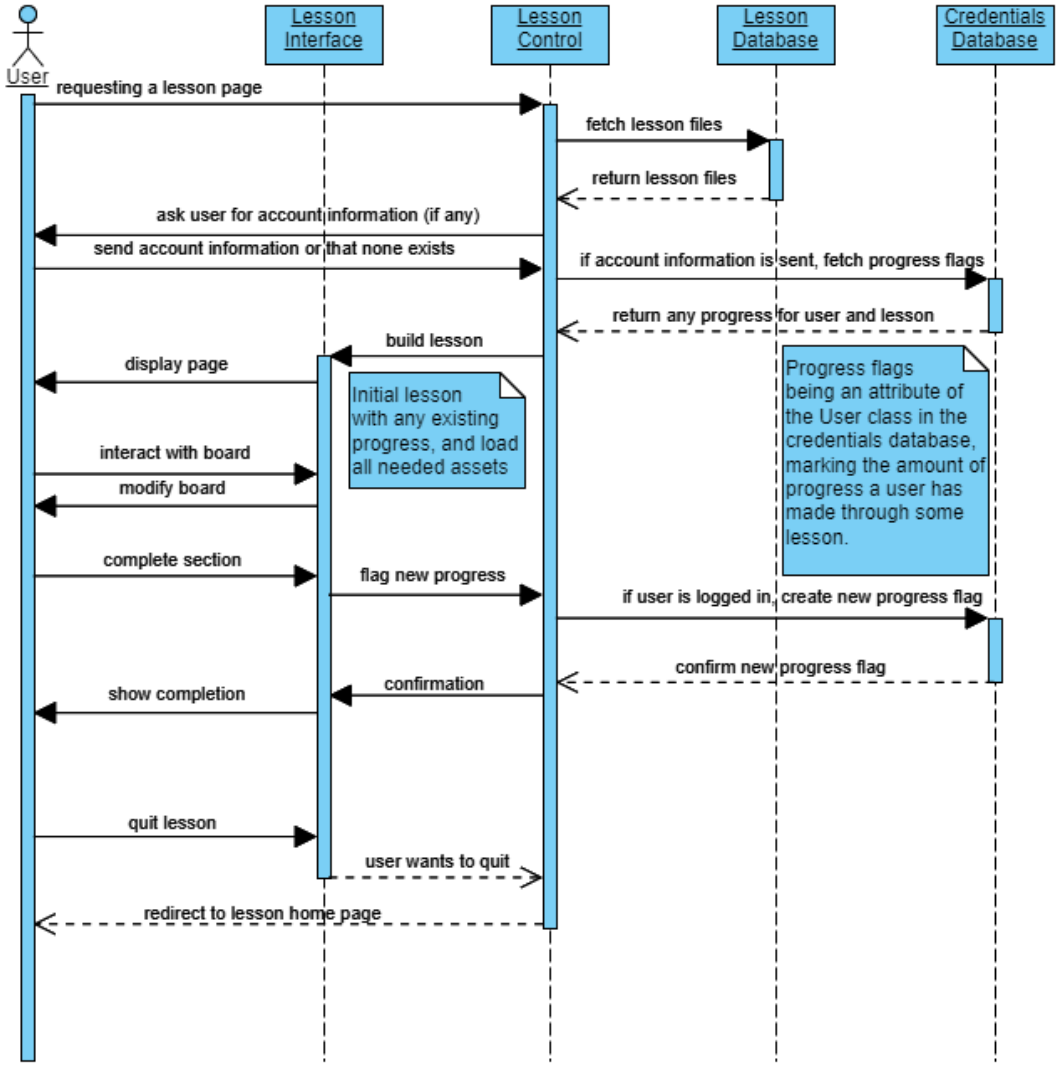
5.1 Flow of Events - Design

An authenticated user navigates to a course's page. If the user has previous progress, the progress data is retrieved and loaded to allow continuation from the user's previous stopping point. Otherwise, a new course is loaded for the user to begin.

5.2 Interaction Diagrams

5.2.1 Sequence Diagrams

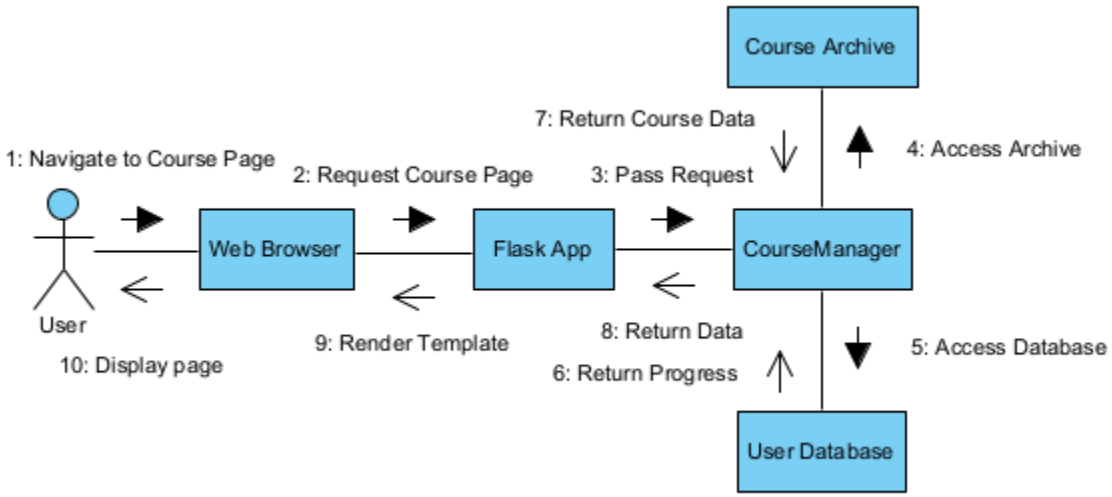
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Load Course>.



[Figure 7: Sequence Diagram: Load Course]

5.2.2 *Collaboration Diagrams*

This Collaboration Diagram shows the static structure of the Use-Case <Load Course>.



[Figure 8: Collaboration Diagram: Load Course]

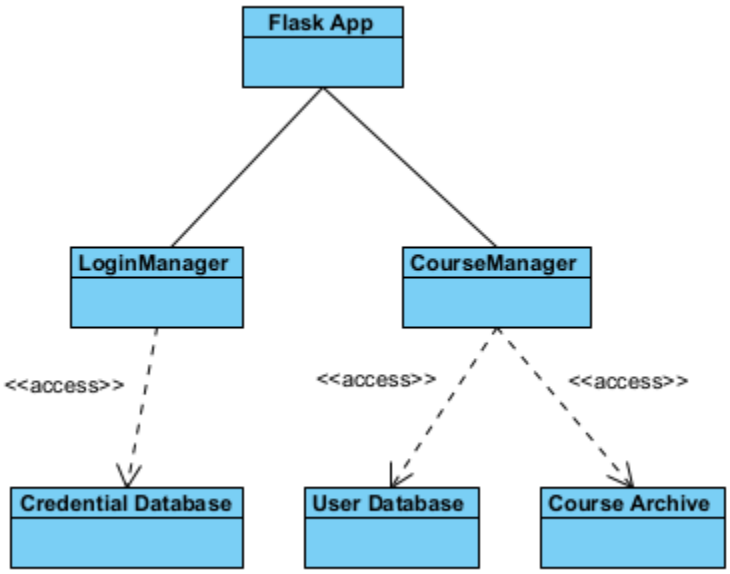
5.2.3 *Participating objects*

The following objects collaborate and define the Use-Case <Load Course>:

Web Browser	This object represents the tool used to access the system and the visible part of the application.
Flask App	This object handles HTTP methods and requests and acts as the system's REST API to load web browser pages and redirect the user.
CourseManager	This object makes queries towards the Course Archive to retrieve file paths to course html pages, and the User Database to retrieve course progress data.
Course Archive	This object receives SQLite queries to provide the full path to a requesting course page.
User Database	This object receives SQLite queries to retrieve tracked progress for a user's course.

5.3 Class Diagrams

The following Object Diagram shows the relations and constraints between Classes and Objects involved in the Use-Case.



[Figure 9: Class Diagram: Load Course]

5.4 Derived Requirements

- If a user has made previous progress in a course, their progress should be retrieved and displayed the next time the user navigates to the same course.

6. USE CASE <Local Game>

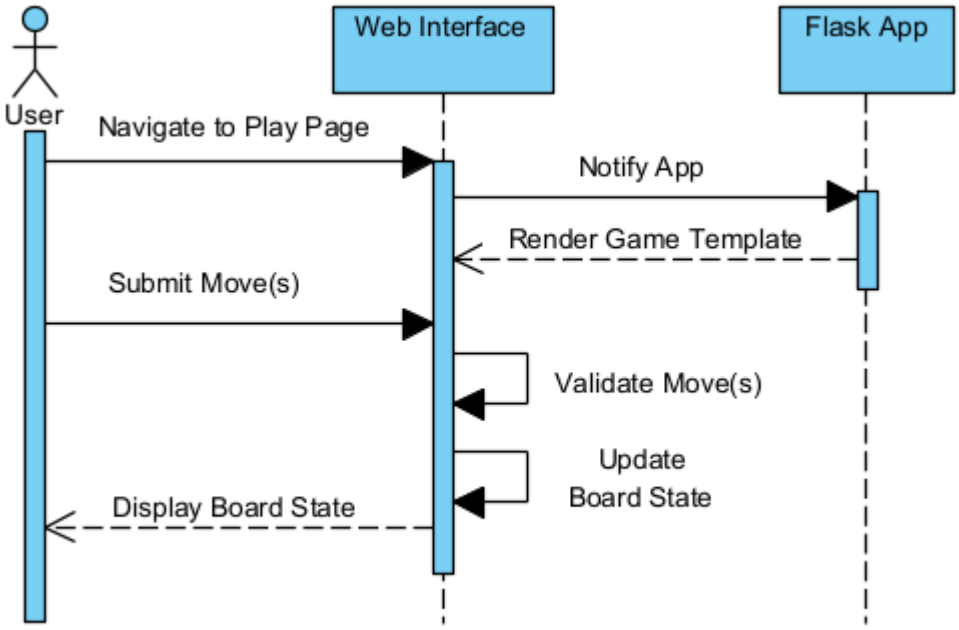
6.1 Flow of Events - Design

A user navigates to the practice page to start a new local game of chess. The user and another local player should be able to submit moves through the web browser. The Flask App will then validate the given moves and update the displayed board state accordingly.

6.2 Interaction Diagrams

6.2.1 Sequence Diagrams

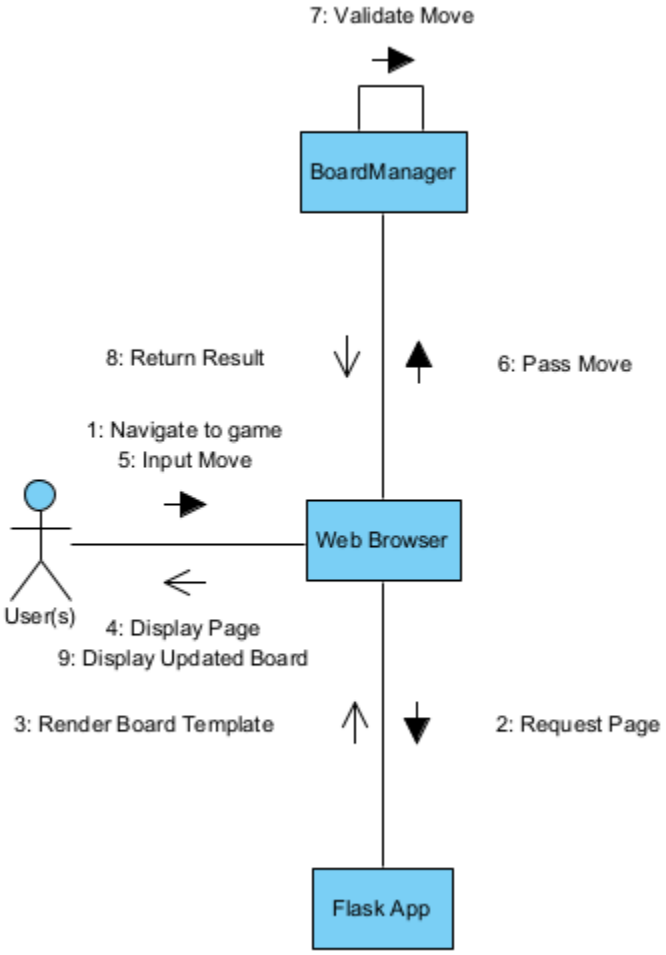
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Local Game>.



[Figure 10: Sequence Diagram: Local Game]

6.2.2 *Collaboration Diagrams*

This Collaboration Diagram shows the static structure of the Use-Case <Local Game>.



[Figure 11: Collaboration Diagram: Local Game]

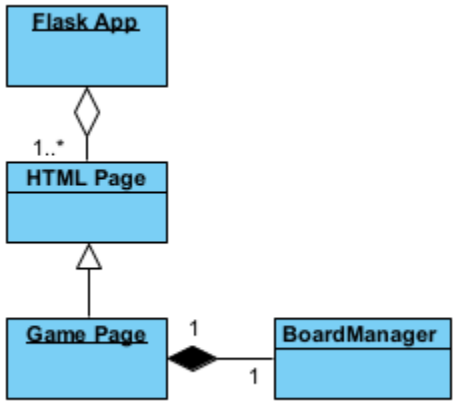
6.2.3 *Participating objects*

The following objects collaborate and define the Use-Case <Local Game>:

Web Brower	This object represents the tool used to access the system and the visible part of the application.
Flask App	This object handles HTTP methods and requests and acts as the system's REST API to load web browser pages and redirect the user.
BoardManager	This object generates a predetermined chessboard and updates the board accordingly after being given a move to process.

6.3 Class Diagrams

The following Object Diagram shows the relations and constraints between Classes and Objects involved in the Use-Case.



[Figure 12: Class Diagram: Local Game]

6.4 Derived Requirements

- The BoardManager object should have a method for creating a full chessboard to make calls for a local game simpler.

LearningEDU

ChessEDU
Iteration Plan <Iteration ID>

Version <1.4>

Revision History

Date	Version	Description	Author
1/11/2022	1.0	First Draft	Rylan DeGarmo
4/11/2022	1.1	Revisions and finalization	Adair Torres
10/11/2022	1.2	Addressed TA's comments	Grant Jones
11/11/2022	1.3	Revised for Iteration 2	Adair Torres
11/16/2022	1.4	Revised for Iteration 3	Adair Torres

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Scope 4
   - 1.3 Definitions, Acronyms, and Abbreviations 4
   - 1.4 References 4
   - 1.5 Overview 4

2. Plan **Error! Bookmark not defined.**

3. Resources 5
   - 3.1 Human Resources 5
   - 3.2 Software Resources 5
   - 3.3 Hardware Resources 5

4. Use Cases 6

5. Evaluation Criteria 6

Iteration Plan <Iteration ID>

1. Introduction

1.1 Purpose

In this third iteration, the team's mission is to develop a final version of the ChessEDU system. All of ChessEDU's functionalities must be implemented and operational. All graphical user interfaces must be able to fulfill their full functionalities.

1.2 Scope

This plan targets tasks and activities assignments especially in relation with artifacts production and code breakdown. This plan offers a vision on how these tasks and activities will be assigned among team members and what roles are involved during this iteration:

Implementer(s)

The implementer(s) participate to the following activities:

- Implement all graphical user interfaces for the web browser pages rendered by Flask.
- Implement Course Browser pagination features and redirection to selected course pages.
- Optimize code to improve efficiency and reduce load times.

Related Artifacts are: Code (Build)

Integrator

The Integrator participates to the following activities: Integrate the system and subsystems (Flask REST API, Manager objects, and databases).

Tester

The tester participates to the following activities: Plan tests for system functionalities and for all graphical interface(s) interactions.

Project Manager

The project manager participates to the following activities: Plan phases and iterations, develop iteration plan, schedule and assign work.

Related Artifacts are: Iteration Plan

1.3 Definitions, Acronyms, and Abbreviations

Refer to the Glossary Document (See References)

1.4 References

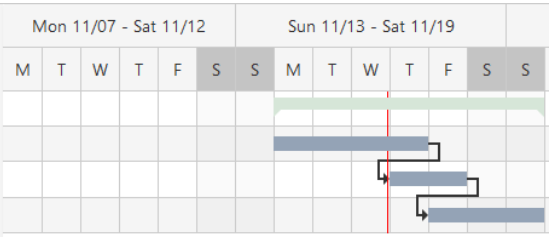
- Glossary Document, Glossary, LearningEDU, 2022
- The course web page https://people.eecs.ku.edu/~saiedian/Teaching/448/

1.5 Overview

This document presents the planning for the iteration and all resources needed.

2. Plan

ID	Title	Start Time	End Time
1	Iteration 3	11/14/2022	11/21/2022
2	Course Browser	11/14/2022	11/18/2022
3	Page Styling	11/17/2022	11/19/2022
4	Code Optimization	11/18/2022	11/21/2022



[Figure 2.1, Iteration 3 Gantt Diagram]

3. Resources

3.1 Human Resources

- *Project's Team:* Adair Torres, Chinh Nguyen, Jack Reynolds, Grant Jones, Rylan DeGarmo
- All course members: professors and lab assistant.

3.2 Software Resources

- Microsoft Office 365 Student
- Google Chrome
- Mozilla Firefox

3.3 Hardware Resources

- The course labs.
- Personal computers.

4. Use Cases

Iteration-Related Use-Cases:

- Course Browser

5. Evaluation Criteria

- Functionality:
  - A page for the user to browse and select a course must be available and fully functional. Hyperlink anchors must be properly loaded and lead to the correct page.
  - Design / styling for all pages must be finalized.
  - Code is optimized to maximize efficiency and reduce page load times.
- Performance:
  - Web pages must load on all test client devices within a timeframe of 10 seconds.
  - Small volumes of requests do not cause services to slow dramatically or stop.
  - Minimal amounts of data should be collected from the user outside of their credentials and progress.
  - The system must operate within 8GB of RAM and 100GB of storage memory.