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

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

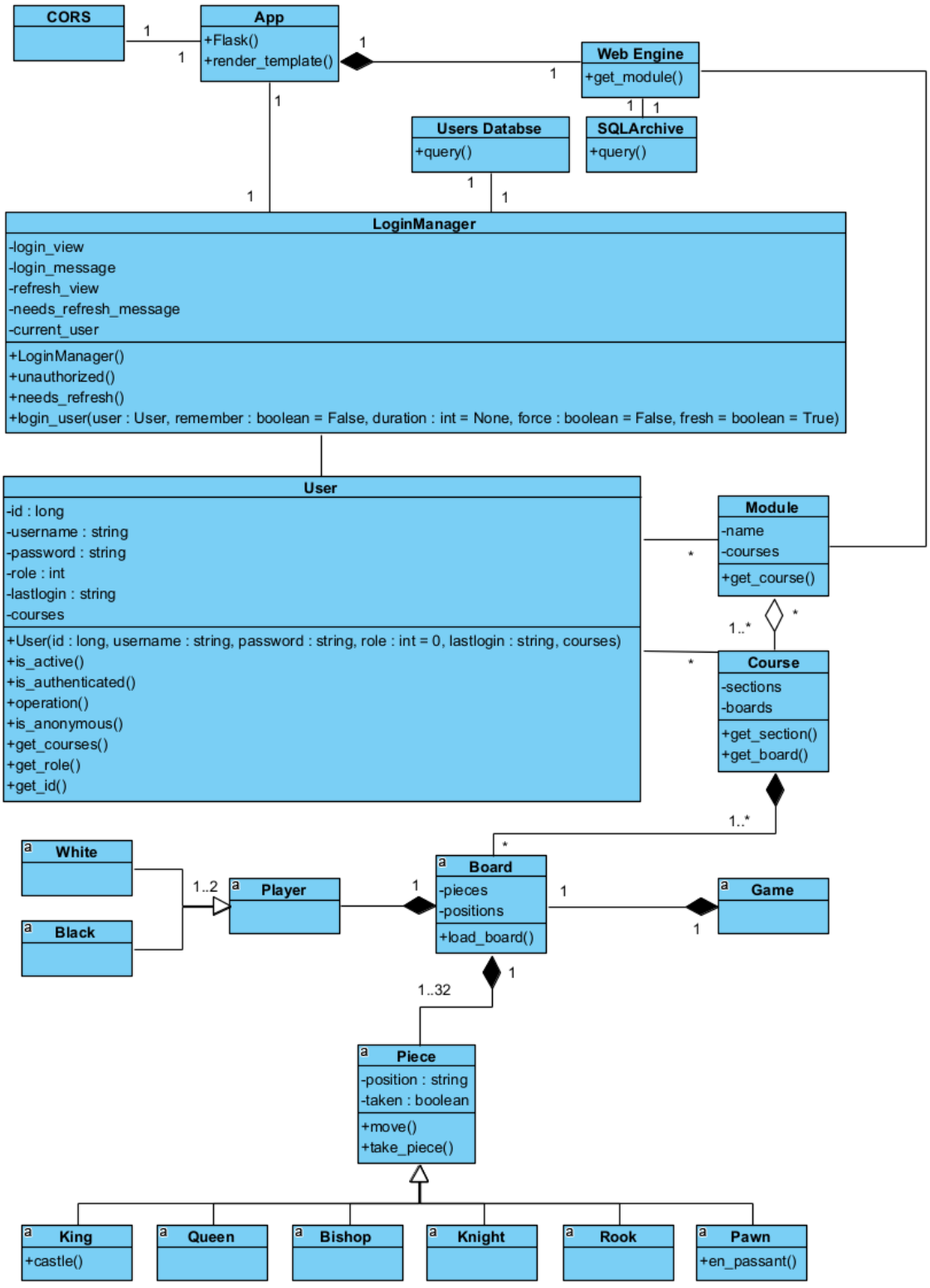
This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities.

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.