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

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>

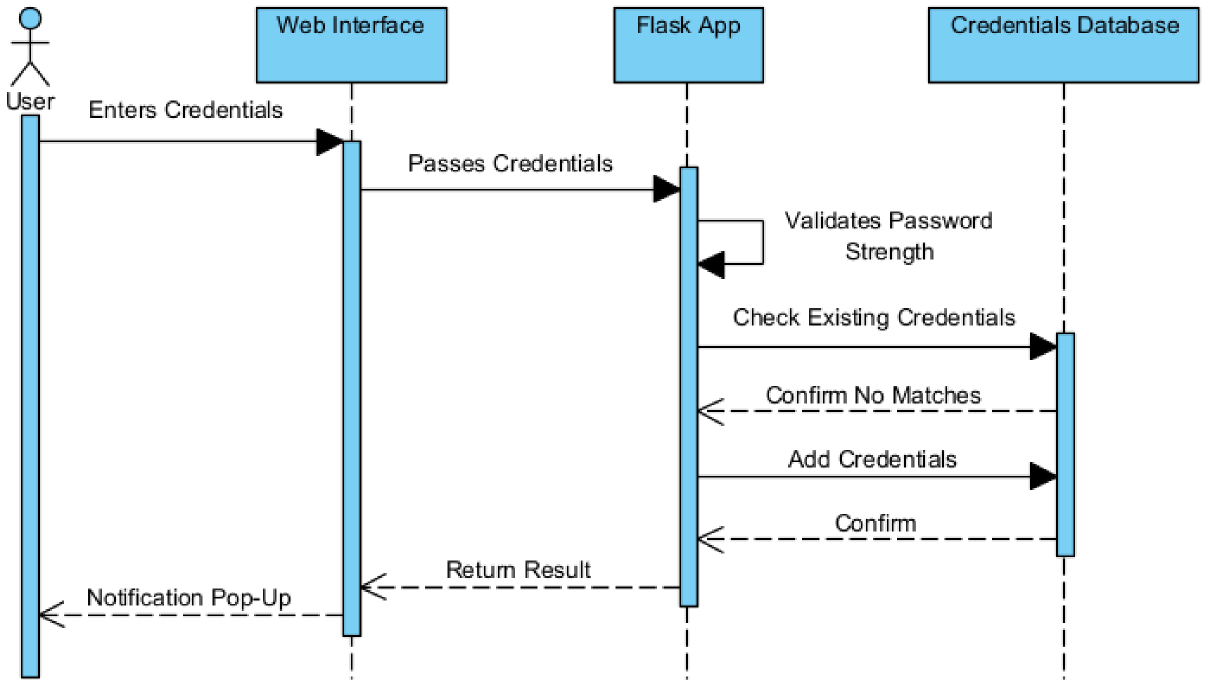
2.1 Flow of Events - Design

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

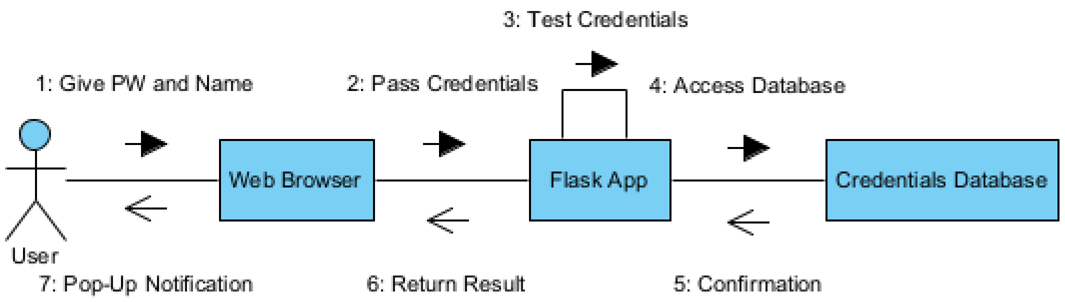
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Sign Up>.



[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

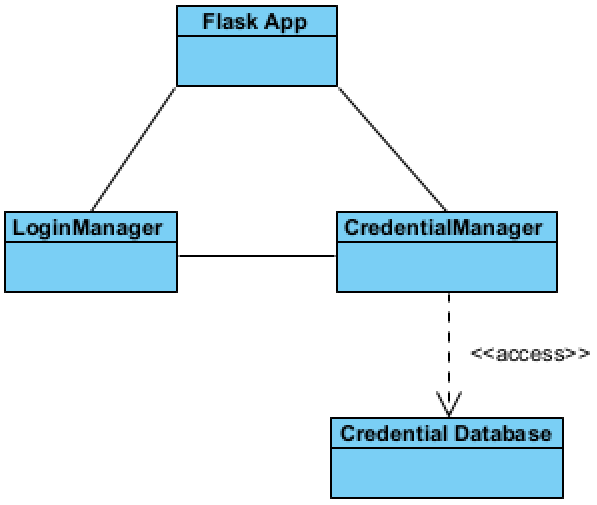
2.2.3 *Participating objects*

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>

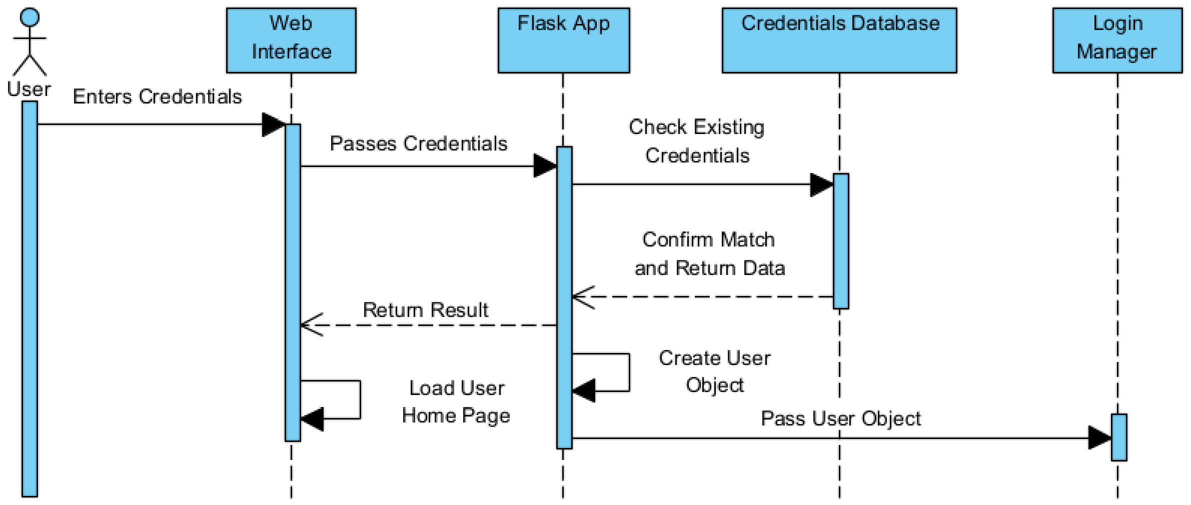
3.1 Flow of Events - Design

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

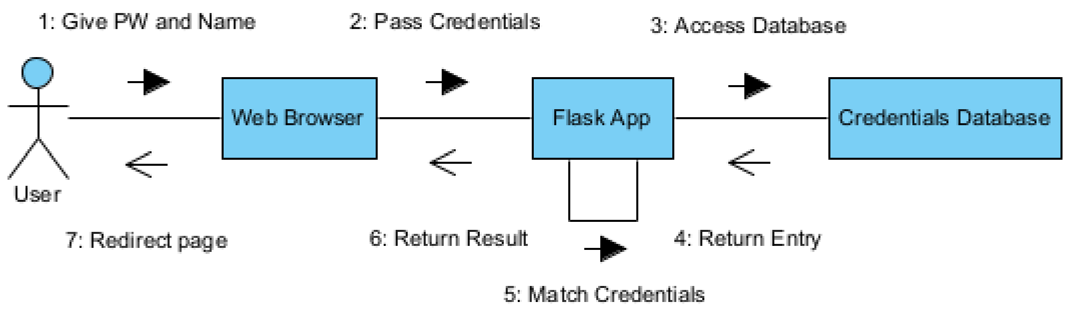
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Login>.



[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]

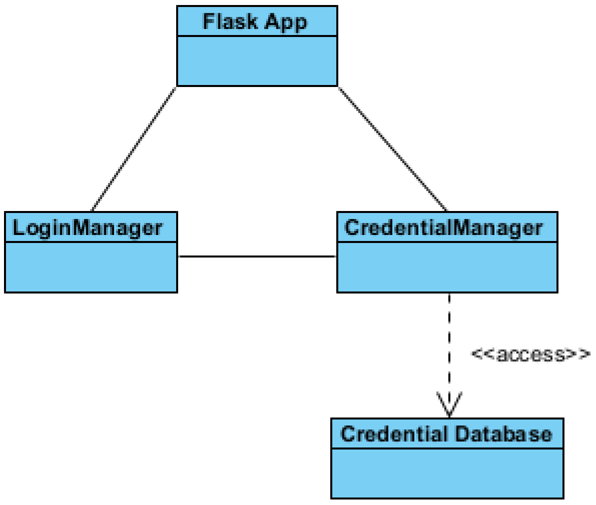
3.2.3 Participating objects

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>

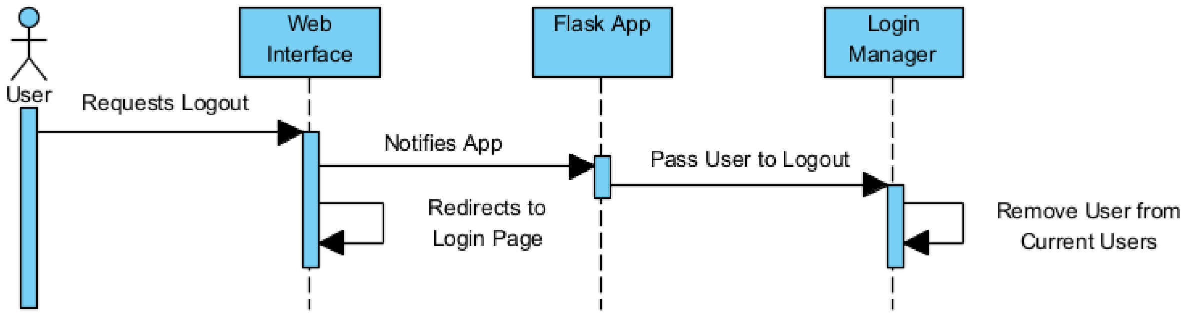
4.1 Flow of Events - Design

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

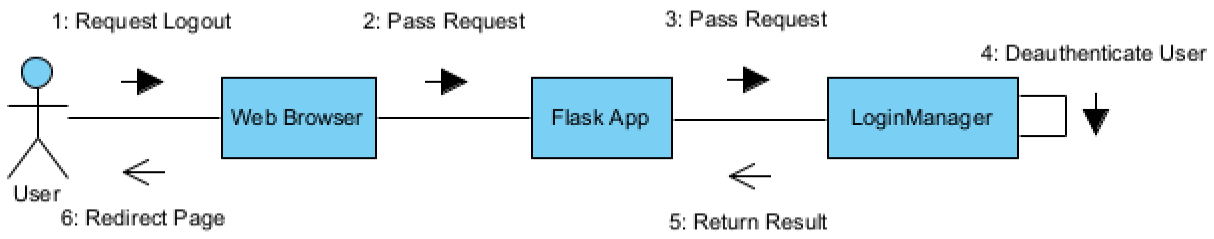
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Logout>.



[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]

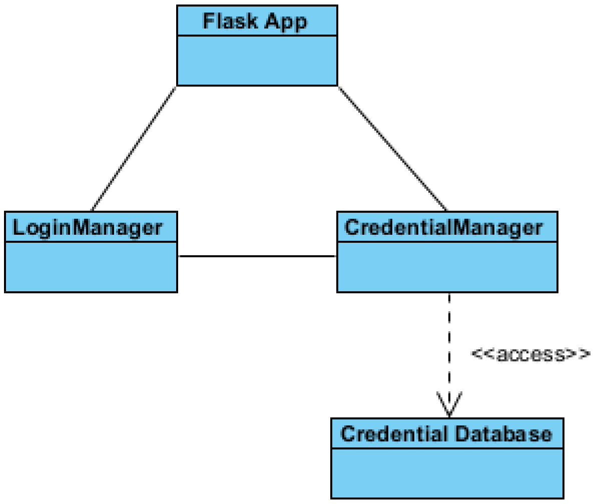
4.2.3 Participating objects

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>

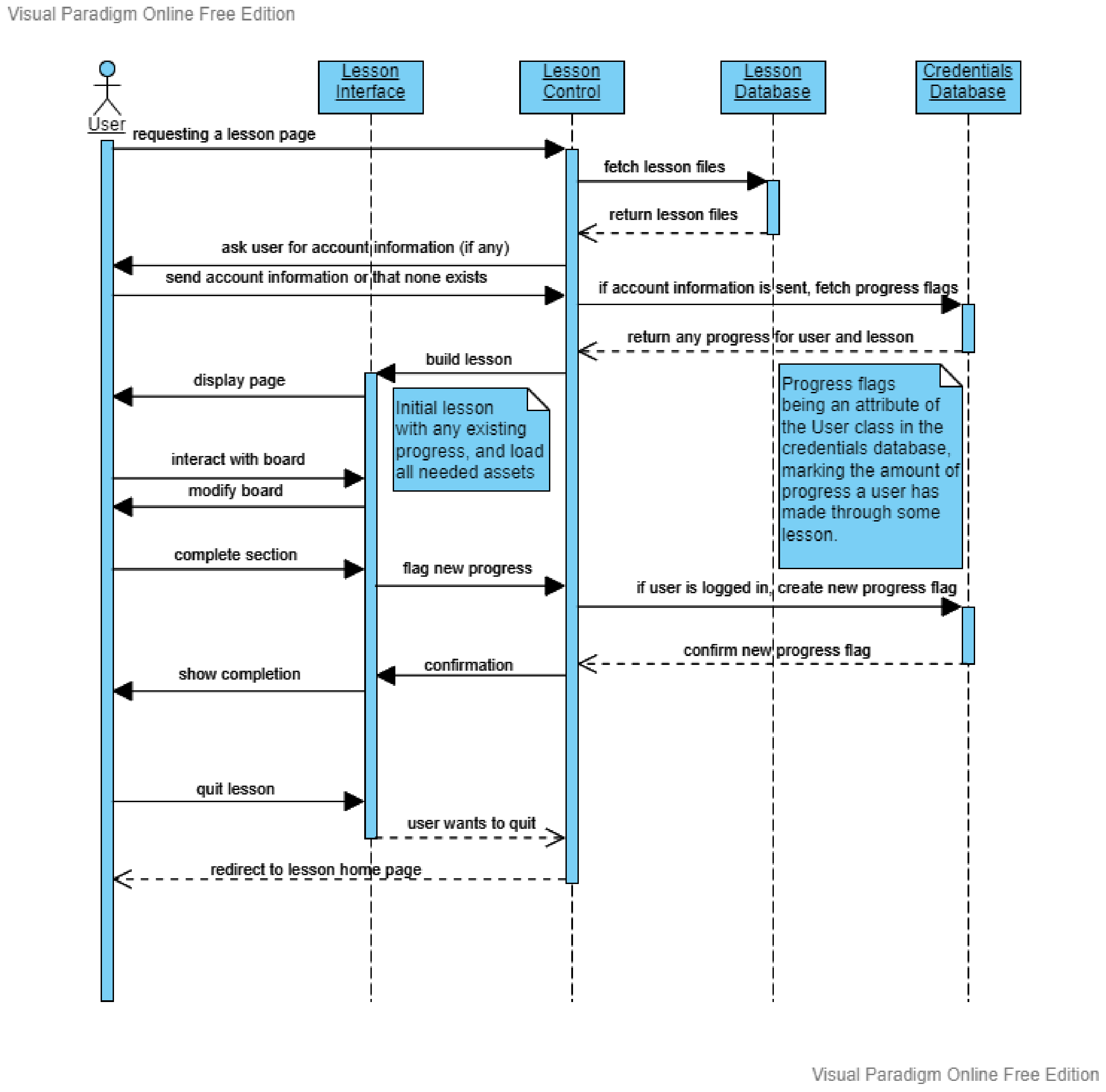
5.1 Flow of Events - Design

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

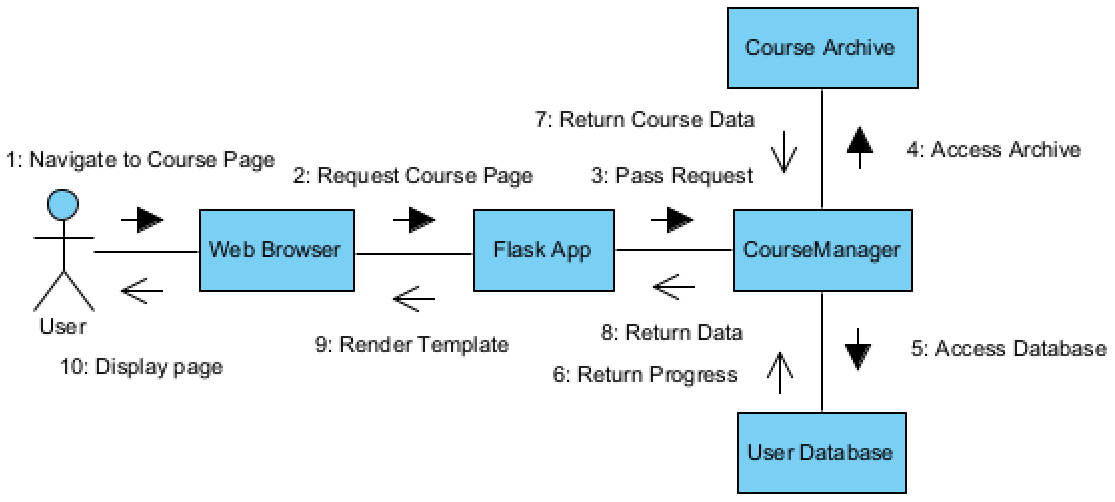
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Load Course>.


Visual Paradigm Online Free Edition
User
Lesson Interface
Lesson Control
Lesson Database
Credentials Database
requesting a lesson page
fetch lesson files
return lesson files
ask user for account information (if any)
send account information or that none exists
if account information is sent, fetch progress flags
return any progress for user and lesson
build lesson
display page
Initial lesson with any existing progress, and load all needed assets
Progress flags being an attribute of the User class in the credentials database, marking the amount of progress a user has made through some lesson.
interact with board
modify board
complete section
flag new progress
if user is logged in, create new progress flag
confirm new progress flag
confirmation
show completion
quit lesson
user wants to quit
redirect to lesson home page
Visual Paradigm Online Free Edition

[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]

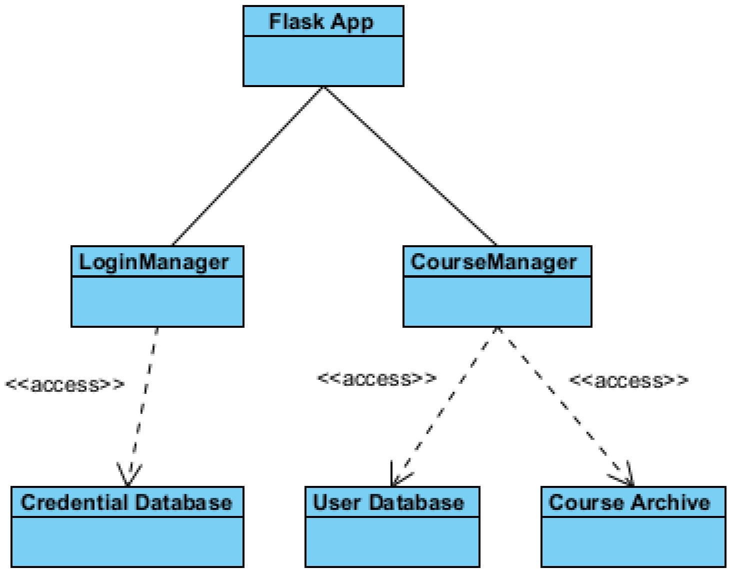
5.2.3 Participating objects

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>

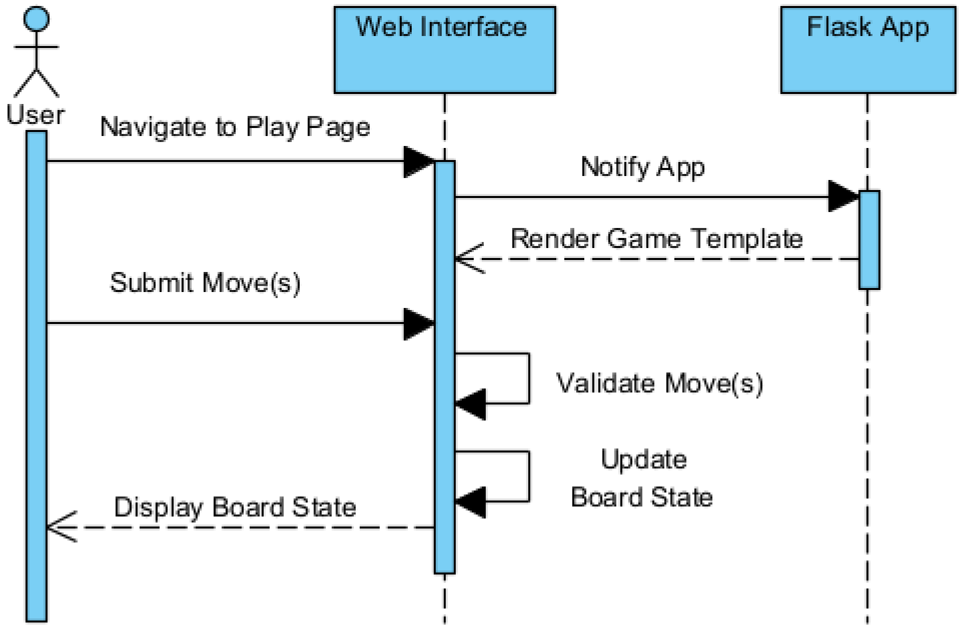
6.1 Flow of Events - Design

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

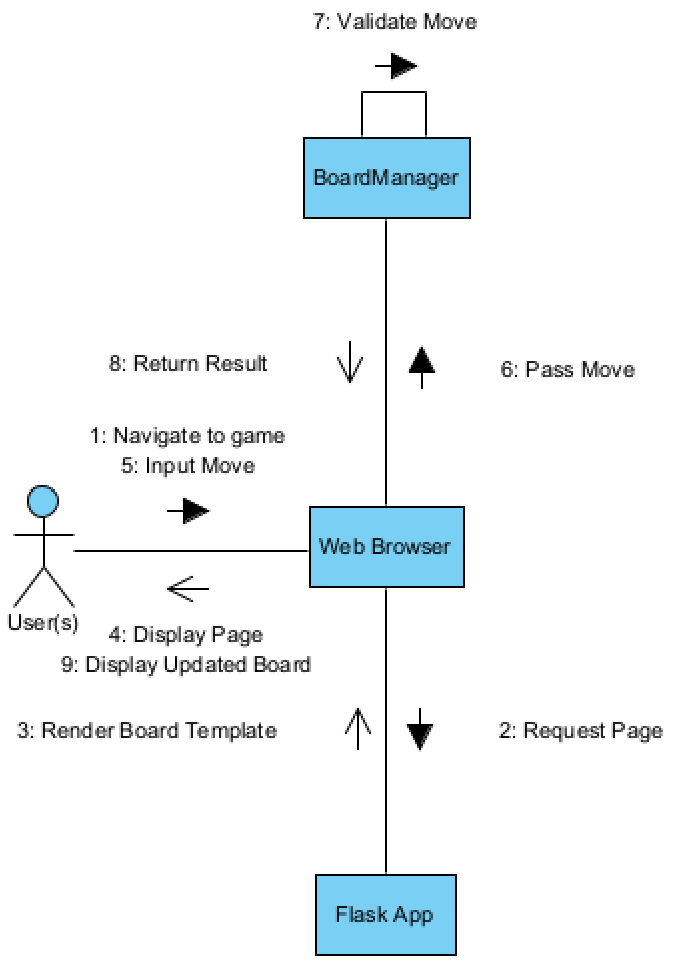
This Sequence Diagram shows Actors and Objects exchange messages in the Use-Case <Local Game>.



[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]

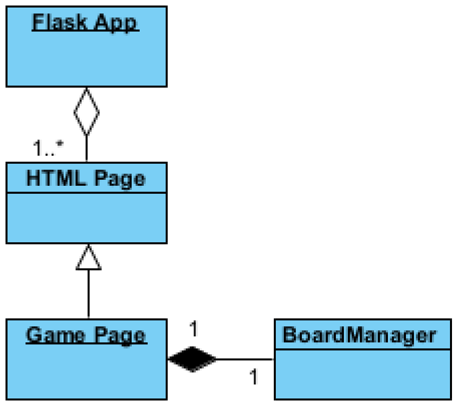
6.2.3 *Participating objects*

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.