Acknowledgments

I, **Sami Kechiche**, would like to express my sincere appreciation to all those who have contributed, directly or indirectly, to the successful completion of this Integration Project. This work marks an important stage in my academic journey, and it would not have been possible without the support, guidance, and encouragement of many individuals.

First and foremost, I extend my deepest gratitude to my supervising teachers, **Mr. Firas Chouchene Hamila** and **Ms. Imen Ferchichi**, whose expertise, constructive feedback, and patience have been invaluable throughout the development of this project. Their commitment to academic excellence and their willingness to provide clear direction played a significant role in shaping both the technical and conceptual aspects of my work.

I would also like to acknowledge the faculty members of my institution for cultivating a learning environment that continuously motivates students to grow, innovate, and push the limits of their abilities. Their dedication has enriched my understanding of computer science and encouraged me to explore real-world problem-solving beyond theoretical coursework.

My sincere thanks go to my family for their unwavering support. Their constant encouragement, understanding, and belief in my potential have been a source of strength throughout this project. I am especially grateful for their patience during the long hours I dedicated to research, design, and development.

I am also thankful to my friends and classmates who offered insightful discussions, shared resources, and stood by me throughout the project. Their collaboration and moral support contributed greatly to maintaining my motivation and ensuring steady progress.

Finally, I would like to acknowledge the broader community of developers and educators who share knowledge openly through documentation, tutorials, forums, and open-source tools. The accessibility of these resources played a crucial role in overcoming challenges and refining the technical components of this project.

To everyone who contributed in any way, whether mentioned explicitly or not, I express my sincere gratitude. This project stands as much on your support as it does on my efforts.

Table of Contents

List of Figures

List of Tables

General Introduction

In recent years, digital learning tools have significantly transformed the way students acquire, retain, and review knowledge. Among these tools, flashcard-based learning systems have become particularly popular due to their effectiveness in reinforcing memorization through active recall and spaced repetition. Despite this growing adoption, many existing applications remain constrained by unnecessary complexity, limited customization, or user interfaces that do not provide a sufficiently simple and focused study experience.

This project is carried out as part of the Integration Project for the first semester of the fourth year in Software Engineering at **EPI Digital School, Sousse, Tunisia**. Its main objective is to apply the concepts and technologies learned throughout the semester by developing a functional software application. In this context, the chosen solution, **MemoryDeck**, is a lightweight web platform that allows learners to create, organize, and practice digital flashcards. The application focuses on simplicity, usability, and clarity, aiming to offer learners an accessible tool that supports efficient and personalized revision.

This report presents the methodological approach and the development process adopted throughout the project. It is organized into three main chapters:

- **Chapter 1** presents the preliminary study, including the project context, motivations, institutional environment, analysis of existing solutions, and a detailed specification of functional and non-functional requirements.

- **Chapter 2** outlines the conceptual design of the system through UML modeling, including the use case diagrams, class diagram, and sequence diagram.

- **Chapter 3** details the implementation phase, including the chosen technologies, system architecture, and the main interfaces of the developed application.

Finally, the report concludes by summarizing the results achieved and highlighting potential improvements.

Chapter 1 — Preliminary Study and Requirement Specification

Chapter 1 — Preliminary Study and Requirement Specification

Introduction

This first chapter establishes the foundations of the project by presenting its general context, motivations, and the institutional environment in which it has been carried out. It then provides an analysis of existing solutions, highlighting their strengths and limitations, followed by a clear and detailed specification of the functional and non-functional requirements that guide the design and implementation of the proposed application. Finally, it outlines the development methodology adopted for the project.

1.1. Presentation of the Host Institution

EPI Digital School is a modern higher-education institution established within the EPI Group's Grand Campus in Sousse, Tunisia, which opened its doors in the 2022-2023 academic year. Its programs span preparatory cycles, bachelor's degrees, master's degrees, and engineering tracks in various IT and digital domains such as Software Engineering, Artificial Intelligence, Cybersecurity, IoT, Big Data, Cloud Computing, and Digital Marketing. Equipped with over 40 specialized laboratories, a Data Center, certification facilities, and high-performance connectivity, the school provides an environment that supports applied learning and real-world projects. Its educational mission aligns with international standards, notably through adherence to the **EURO-INF** accreditation frameworks. This project fits within the institution's vision of training engineers capable of designing practical and innovative digital solutions [1].

Figure 1: EPI Digital School

1.2. General Context and Motivation

This section presents the academic framework of the project and explains the motivations behind the creation of MemoryDeck. It highlights the relevance of such a tool in today's learning environment and how it aligns with the objectives of the Integration Project.

1.2.1. Project Framework

This project is carried out as part of the Integration Project for the first semester of the fourth year in Software Engineering at EPI Digital School, Sousse, Tunisia, for the academic year 2025–2026. It aims to provide students with an opportunity to apply their theoretical knowledge by designing and developing a functional software application, while also preparing them for larger-scale projects such as the PFA and PFE. Within this context, the objective is to conceive and implement **MemoryDeck**, a web-based application dedicated to creating, organizing, and practicing personalized study flashcards.

1.2.2. Problem Description and Motivation

Students commonly prepare for exams using printed notes, summaries, and old test sheets. However, this traditional approach presents several limitations:

- Materials are often **lost, damaged, or incomplete**.
- Students tend to memorize the fixed order of questions instead of the underlying content.
- Physical materials are not accessible across devices.
- No performance history is tracked, making progress difficult to evaluate.

Digital study tools address many of these limitations by:

- Centralizing all revision materials in one place
- Allowing reordering or randomization for improved learning
- Ensuring accessibility from any device with a browser
- Providing feedback and tracking performance

Unfortunately, many existing platforms remain overloaded with features, locked behind paywalls, or insufficiently intuitive for learners seeking a simple, distraction-free solution. This motivates the creation of **MemoryDeck**, a lightweight flashcard application that enables learners to create and practice their study materials quickly and efficiently, without unnecessary complexity.

1.3. Study of Existing Solutions

This section presents a preliminary study of well-known digital flashcard systems. The goal is to analyze the strengths and limitations of each solution in order to position

MemoryDeck appropriately and avoid repeating the drawbacks commonly found in the existing tools.

For this project, three representative tools were selected: **Anki**, **Quizlet**, and **Kahoot**. They were chosen because they are widely used and represent distinct approaches to learning through memorization.

1.3.1. Anki

Anki is a powerful open-source flashcard program based on spaced repetition algorithms. It allows extensive customization through templates, plugins, and hierarchical deck structures. However, its interface is often considered technical and unintuitive for beginners, resulting in a steep learning curve [2].

Figure 2: Anki Logo

1.3.2. Quizlet

Quizlet is a widely used, user-friendly platform offering flashcards, practice tests, and simple learning modes. Its interface is modern and intuitive. However, many important features—such as offline access, advanced learning modes, and progress tracking—are locked behind a premium subscription, which limits full accessibility for students [3].

Figure 3: Quizlet's Logo

1.3.3. Kahoot

Kahoot is an interactive quiz platform primarily designed mainly for live competitions and classroom environments. It focuses on engagement, game-based learning, and real-time participation. Although highly effective for group-based activities, it is not suited for personal, private, or long-term flashcard study [4].

Figure 4: Kahoot's Logo

1.4. Critique of the Existing Solutions

It is important to critically evaluate the strengths and limitations of existing flashcard learning platforms in order to identify which features our solution should adopt, simplify, or avoid. The following table compares the previously mentioned tools based on key criteria such as purpose, complexity, accessibility, and strengths.

Table 1: Comparison of Existing Learning Tools

Criteria / Platform	**Anki**	**Quizlet**	**Kahoot**
Primary purpose	Long-term spaced-repetition memorization	General study & classroom activities	Live game-based learning, and quizzes for groups
Best for	Individuals seeking deep memorization	Students & teachers wanting quick sets & study modes	Teachers running interactive classroom sessions
Offline use	✓ Fully offline	Limited (offline in premium)	✗ Online only
Registration Required	✗ Optional (local use)	✓ Required for most features	✓ Required
Complexity (learning curve)	High	Low–Medium	Very low
Cost (basic user)	Free (desktop & Android)	Freemium (many key features premium-locked)	Freemium (advanced features paid)
Card richness	Very flexible (cloze, images, plugins, templates)	Flexible, but simpler	Limited to MCQ-style content
Strengths	Powerful algorithm; highly customizable	User-friendly, polished, fast creation workflow	Fun, engaging, excellent for live group sessions

Across all three platforms, several common limitations appear:

- Deck or card creation is not always intuitive for beginners.
- Many tools introduce unnecessary complexity for learners who want a simple solution.
- Personal, private study is not always the primary focus.
- Free versions often restrict essential features.

These observations support the relevance of developing a simple, customizable, student-oriented flashcard tool such as MemoryDeck.

1.5. Proposed Solution

To address the limitations identified in the previous section, this part introduces MemoryDeck, the solution developed in the context of this project. MemoryDeck is a lightweight web-based application designed to help students create, organize, and practice digital flashcards. It focuses exclusively on the essential features required for effective memorization, prioritizing simplicity, usability, and a clear user interface.

1.6. Requirement specification

Requirements define how the system should behave and what constraints it must follow. This section describes the actor of the system as well as the functional and non-functional requirements.

1.6.1. Identifying the Actors

Before listing the requirements, it is important to identify the main actors who interact with the system:

- **User (Learner):** The user represents any learner who interacts with the system to manage personal decks and cards and to conduct practice sessions.

1.6.2. Functional Requirements

Functional requirements specify what the system must accomplish. Our solution should contain the following requirements:

User Authentication

- The user must be able to register.
- The user must be able to log in using an email and password.
- The user must be able to log out.

Deck Management

- The user can view the list of all decks.
- The user can create a new deck.
- The user can rename an existing deck.
- The user can delete a deck.
- The user can search/filter decks by name.

Card Management

- The user can view all cards that belong to a selected deck.
- The user can create new cards by providing a front and back.
- The user can edit existing cards.
- The user can delete cards.
- Cards belong to exactly **one** deck.

Practice Session

- The user can choose a deck to practice.
- During practice, the user can:
    - Show the next card
    - Reveal the answer
    - Mark answer as **correct**
    - Mark answer as **incorrect**

- Skip card
- End the session at any time

- After the practice session, the system should display the number of correct, incorrect, and skipped cards.

Practice History Management

- The user can view a list of all completed practice sessions.
- The user can filter practice history by deck.
- The user can sort practice history by date.
- The user can view the detailed summary of any past practice session.
- The user can delete a single practice session.
- The user can delete all practice history.

1.6.3. Non-functional Requirements

Our solution should contain the following requirements:

- **Performance:** The application must respond quickly and allow smooth navigation, even on basic student hardware.
- **Usability:** The interface must remain simple, intuitive, and accessible to beginner-level users.
- **Maintainability:** The system must be developed using clear code structure and modular design, making future extensions easier to implement.

- **Security:** Passwords must be stored securely using hashing mechanisms, and all operations that modify user data must be accessible only to authenticated users.

These requirements ensure that the system is pleasant and safe to use, not only functionally correct.

1.7. Development Methodology

An appropriate development methodology provides structure and ensures systematic progression through each project phase. This section outlines the methodology adopted for this work.

1.7.1. Adopted Methodology

Given the limited duration and individual nature of the project, a lightweight Iterative-Incremental methodology was adopted. Each iteration includes requirement clarification, mini-design, implementation, testing, documentation, and review. This approach balances structure and flexibility, making it more suitable than a strict Waterfall model and simpler than full Agile frameworks.

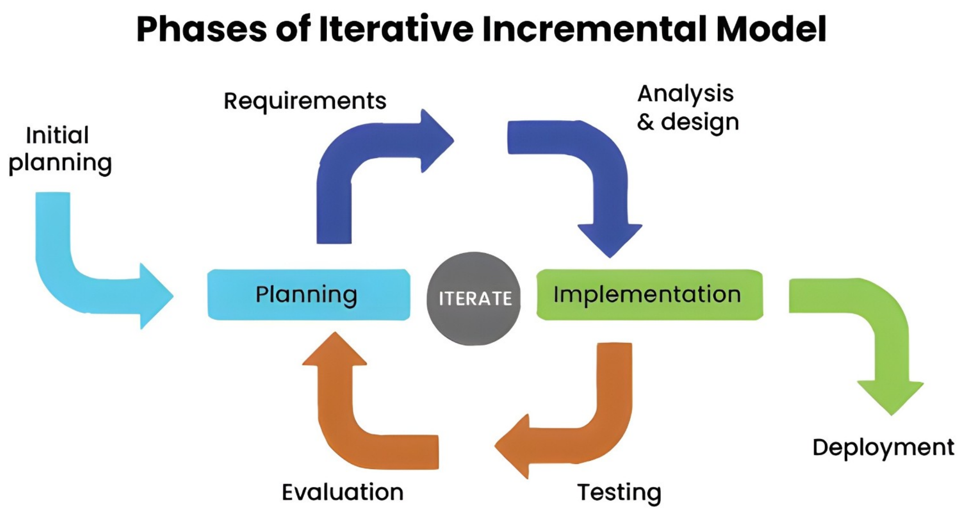


Figure 5: Phases of Iterative Incremental Model

Conclusion

In this chapter, we introduced the general context of the project, the motivations behind MemoryDeck, and the institutional environment in which it was developed. We examined existing flashcard tools to identify their strengths and limitations, proposed an adapted solution, and established the functional and non-functional requirements guiding the project. Finally, the chapter concluded by presenting the development methodology used throughout the work. The following chapter introduces the conceptual design of MemoryDeck through UML diagrams that describe the structure and behavior of the system.

Chapter 2: Conceptual Study

Chapter 2: Conceptual Study

Introduction

This chapter presents the conceptual design of MemoryDeck by transforming the previously defined requirements into a set of structured models based on the Unified Modeling Language (UML). The objective of this conceptual study is to provide a clear understanding of how the system behaves, how its components interact, and how the main functionalities are organized before implementation. The chapter begins by outlining the adopted design methodology, then presents the use case models, including the global use case diagram and the refined diagrams. It continues with the structural and behavioral models, namely the class diagram and the sequence diagram. Together, these models form a coherent conceptual blueprint that guided the implementation of the system.

2.1. Design Methodology

To structure the conception of MemoryDeck, we adopt a **UML-driven design approach**. UML is a standardized object oriented modeling language used to visualize, specify, and document software systems [5]. It offers diagram types that clarify:

- **What the system does** → Use case diagrams
- **How the system is structured** → Class diagrams
- **How the system behaves** → Sequence diagrams

Using UML ensures consistency between requirements and implementation. It helps eliminate ambiguity early, supports modular design, and guides the development of both backend and frontend components. This is especially important for a lightweight academic project where clarity, simplicity, and correctness are essential.

2.2. Use Case Modeling

Use case modeling offers a functional view of *MemoryDeck* by describing the interactions between the system and its user. It highlights the set of features the user can perform, without defining technical implementation details. This section begins with the global use case diagram, which summarizes all system functionalities, and continues with refined diagrams that provide additional detail for the key functional groups.

2.2.1. Global Use Case Diagram

The following global use case diagram presents a high-level overview of all actions available to the user in *MemoryDeck*.

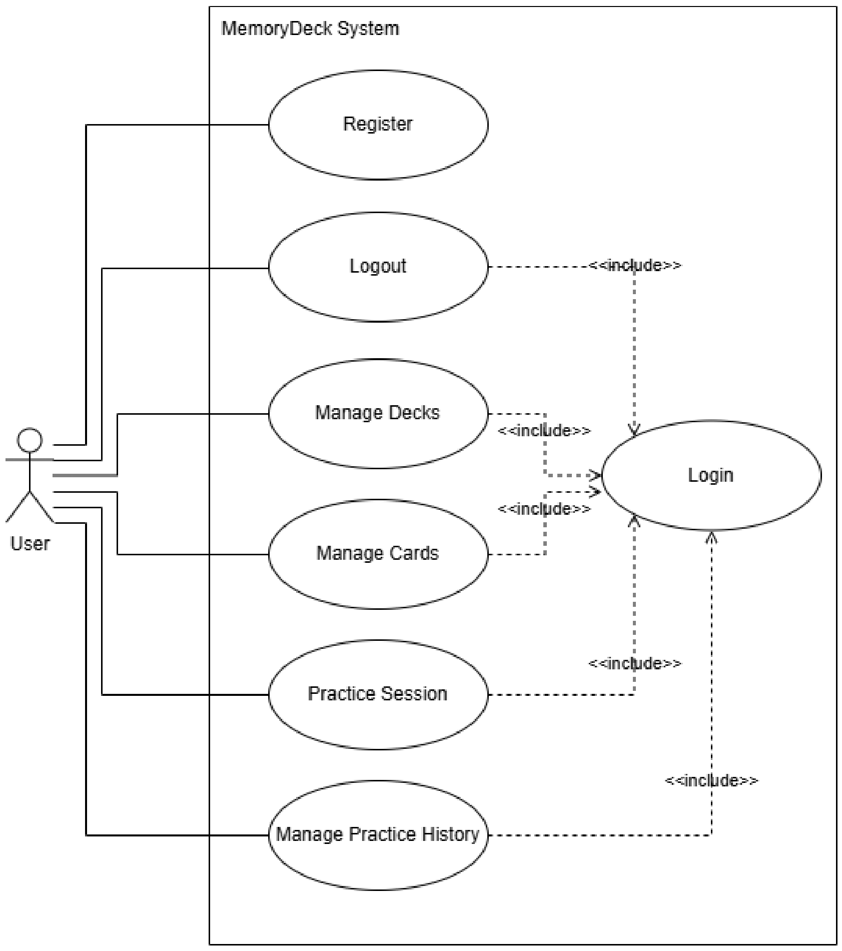


Figure 6: Global Use Case Diagram

2.2.2. Refined Use Case Scenarios

This section presents the refined descriptions of the principal functional components in *MemoryDeck*. Each refined diagram expands a general use case from the global model and explains the specific actions the user may perform within that functional group: deck management, card management, practice sessions, and practice history management.

Refined Use Case 1 — Deck Management

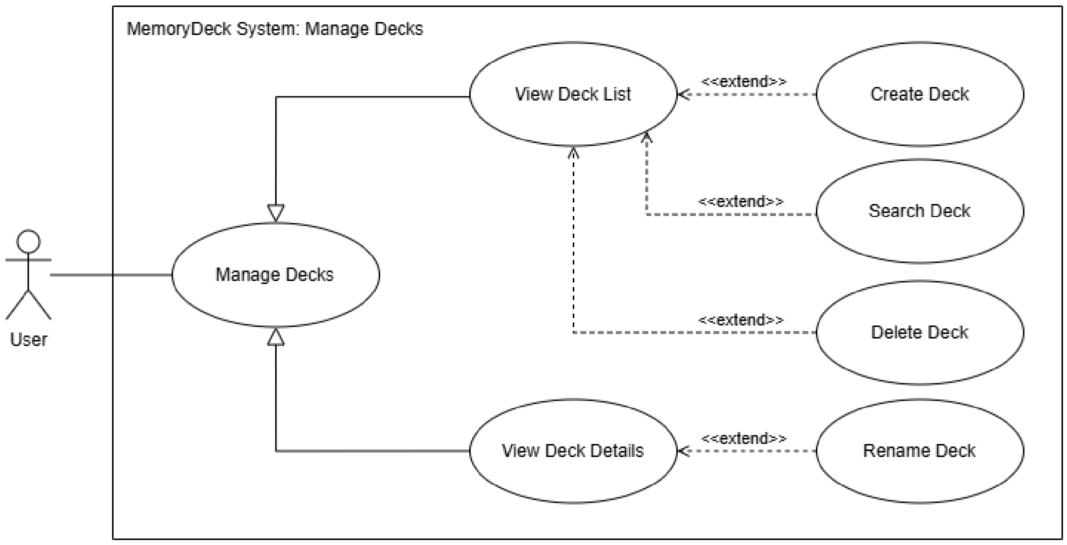


Figure 7: “Deck Management“ Refined Use Case Diagram

This refined use case covers all operations related to organizing and maintaining decks. After successfully authenticating, the user may access the list of all existing decks, search for a specific deck by name, or create a new one. From the deck list, the user may delete a deck or select a deck to view its details. Within the deck details view, the user may rename the deck. These actions allow the learner to structure and maintain their study materials in an organized and consistent manner.

Refined Use Case 2 — Card Management

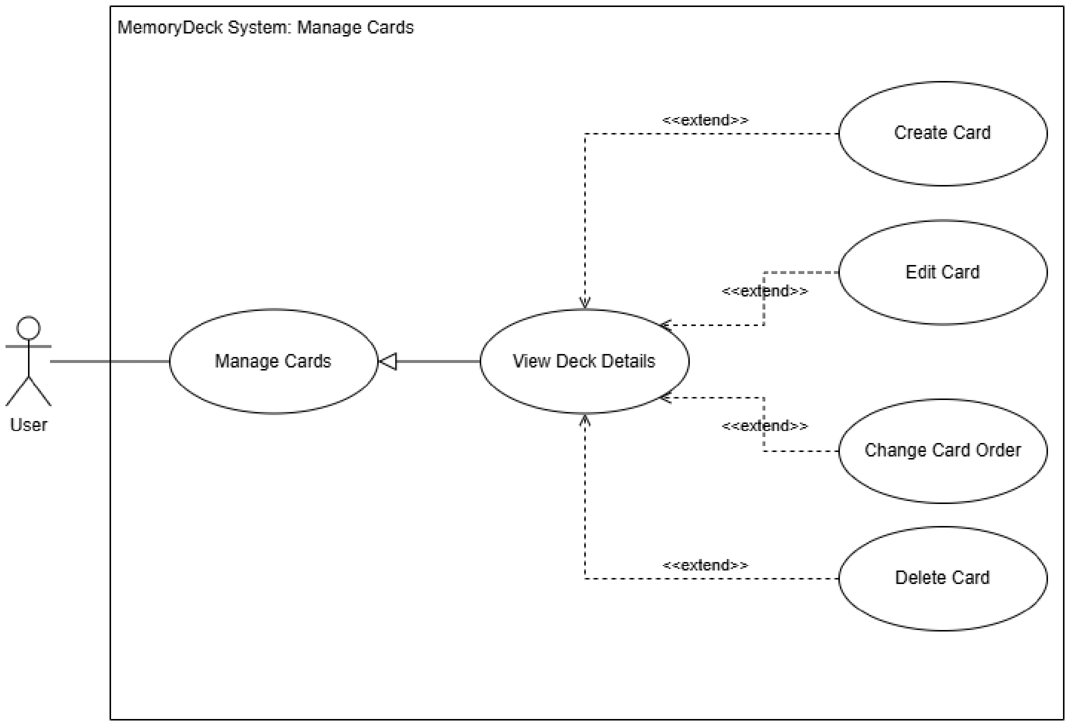


Figure 8: “Card Management” Refined Use Case Diagram

This refined use case focuses on managing the cards contained within a specific deck. When viewing a deck’s details, the user may create new cards by providing front and back content, edit existing cards, delete cards, or change their order within the deck. Cards are always associated with exactly one deck and cannot exist independently. These operations enable the user to progressively build and refine the learning material used during practice sessions.

Refined Use Case 3 — Practice Session

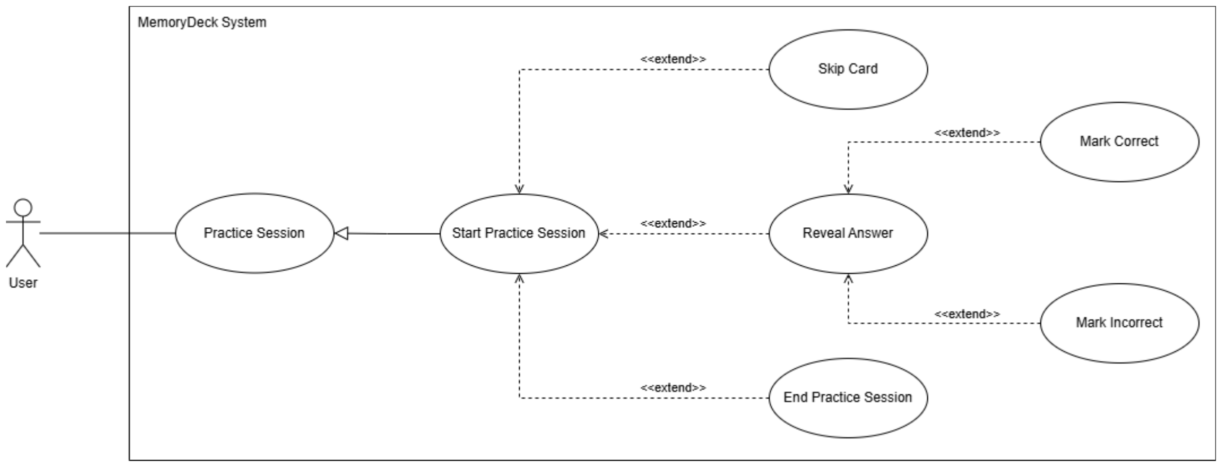


Figure 9: "Practice Session" Refined Use Case Diagram

This refined use case describes the learning workflow supported by MemoryDeck. After selecting a deck, the user may start a practice session. During the session, the system presents cards sequentially based on the order defined in the deck. The user may reveal the answer, mark the response as correct or incorrect, skip the card, or end the session at any time. Once the session ends, the system displays a summary showing the number of correct, incorrect, and skipped cards, along with the session duration. This refined use case represents the core memorization process of the application.

Refined Use Case 4 — Practice History Management

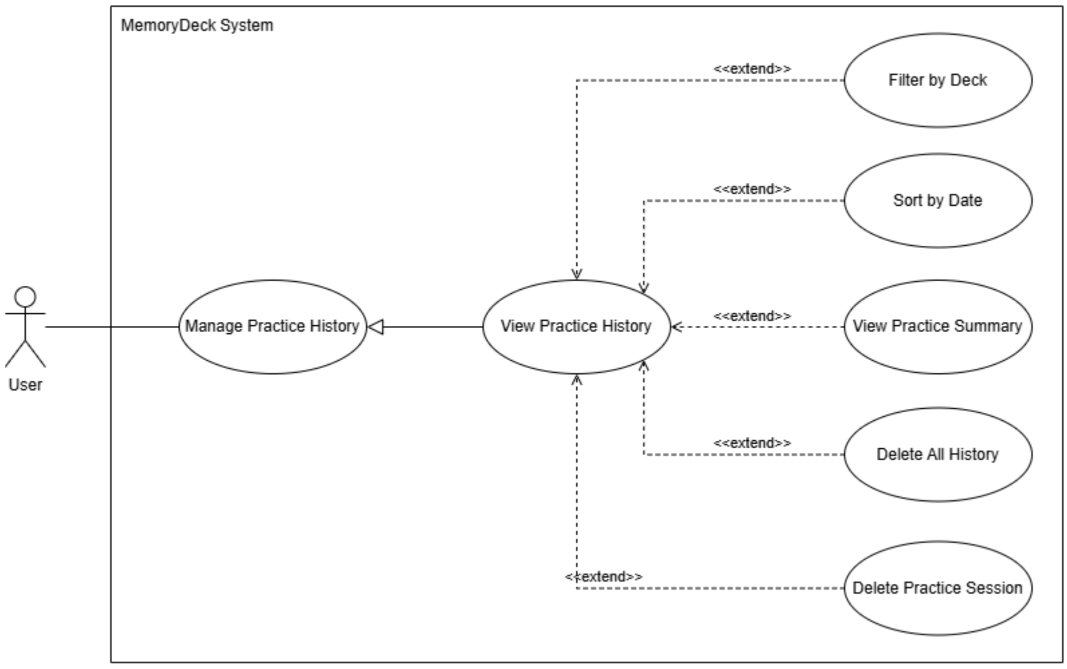


Figure 10: "Practice History Management" Refined Use Case Diagram

This refined use case covers the management of completed practice sessions. The user may access the practice history to view a list of all previously completed sessions across all decks. The history may be filtered by deck and sorted by date. From this list, the user may view the detailed summary of any past session or delete individual sessions or the entire practice history. This use case enables learners to track progress over time and review past performance.

2.3. Class Diagram

The class diagram provides a structural view of the MemoryDeck system by identifying its main classes, their attributes, methods, and the relationships between them.

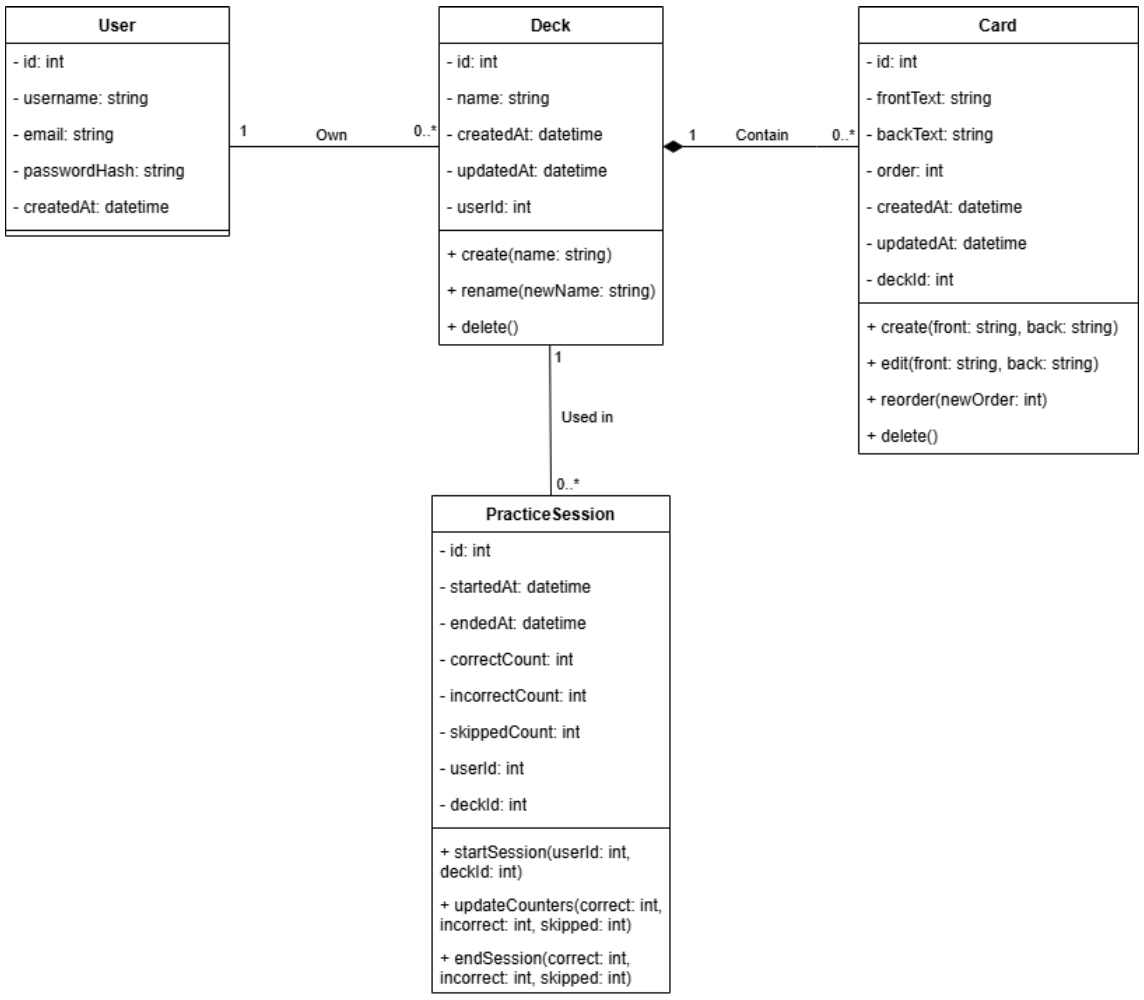


Figure 11: Class Diagram

The model is centered around four main classes: **User**, **Deck**, **Card**, and **PracticeSession**. The User class represents authenticated learners, Deck and Card represent the learning content, and PracticeSession captures the execution and results of study sessions. This class diagram serves as a conceptual foundation for both the database schema and the backend implementation, ensuring coherence between the system's design and its final realization.

2.4. Sequence Diagrams

Sequence diagrams illustrate how different components of the system interact in a chronological order to execute a specific functionality. For MemoryDeck, three sequence diagrams are presented, each corresponding to one of the core functional areas:

- Login
- Card Creation
- Practice Session

2.4.1. Sequence Diagram — Login

The following sequence diagram illustrates the sequence of interactions required for user authentication.

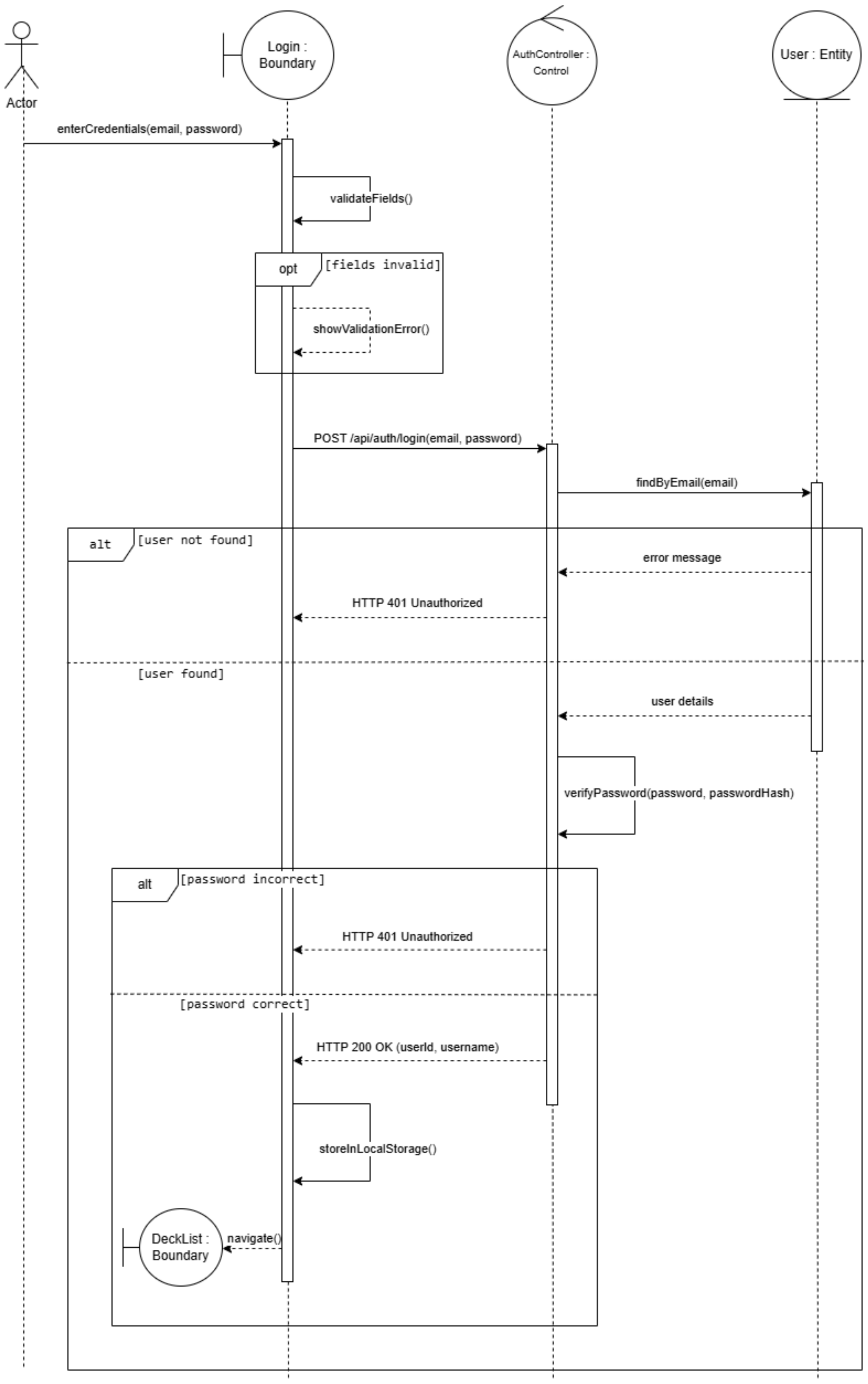


Figure 12: Login Sequence Diagram

To log in, the user enters their email and password through the login interface. A first validation is performed on the client side to ensure that the required fields are filled. If this validation fails, an error message is displayed directly on the interface.
If the input is valid, an HTTP POST request is sent to the authentication controller. The controller queries the user model to verify the existence of a user with the specified email. If the user does not exist or if the password is incorrect, an HTTP *Unauthorized* response is returned to the interface, and an error message is displayed.
If the credentials are valid, the controller returns an HTTP *OK* response containing the user's identifier and username. These values are stored in the browser's local storage, and the user is redirected to the deck list page. This sequence ensures controlled access to the application without exposing sensitive information.

2.4.2. Sequence Diagram — Card Creation

The following sequence diagram illustrates the interactions involved in creating a new card within a deck.

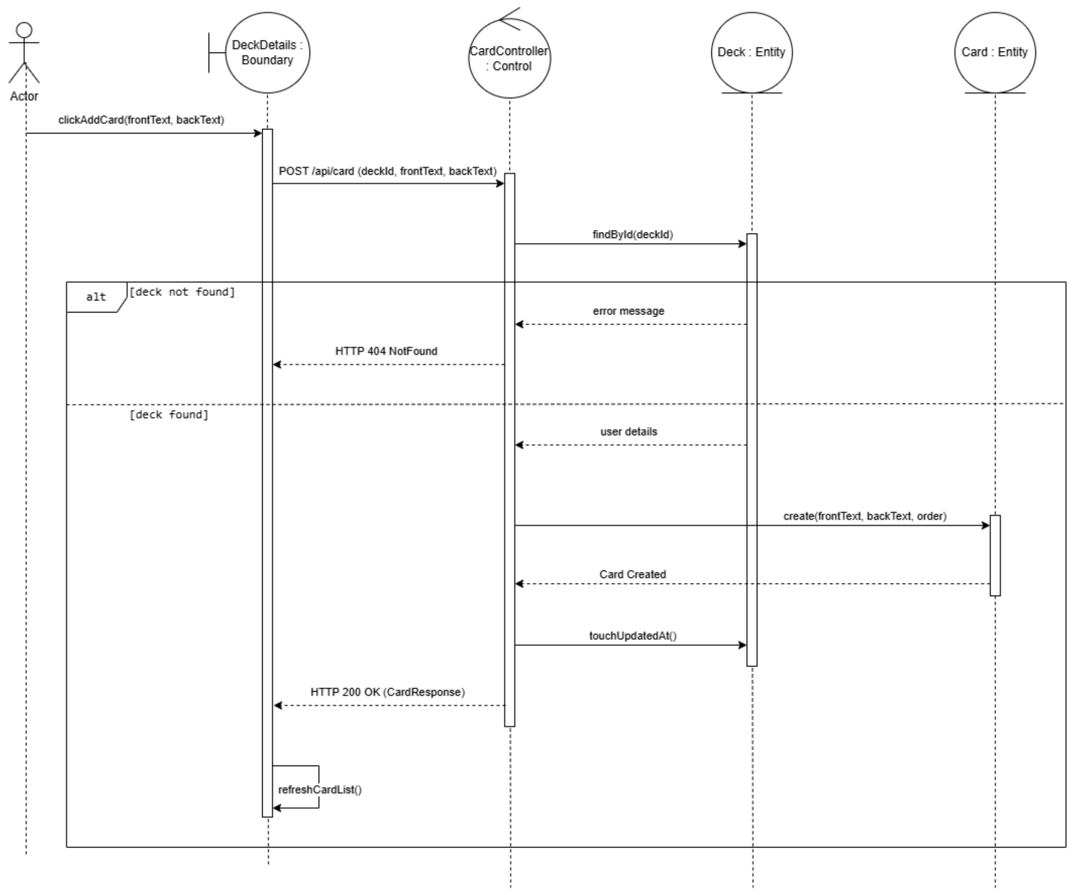


Figure 13: Card Creation Sequence Diagram

To create a card, the user accesses the deck details interface and enters the front and back content of the card. A validation is first performed in the interface to ensure that both fields are filled. If the validation fails, the card creation request is not sent. If the input is valid, an HTTP POST request is sent to the card controller. The controller verifies the existence of the associated deck by querying the deck model. Once the deck is confirmed, a new card is created and stored in the database with the appropriate order value. After successful creation, the controller returns an HTTP *OK* response containing the created card's data. The interface then refreshes the card list and updates the deck's information. This sequence guarantees that cards are always created within an existing deck.

2.4.3. Sequence Diagram — Practice Session

The following sequence diagram describes the execution of a practice session, which represents the core learning workflow of MemoryDeck.

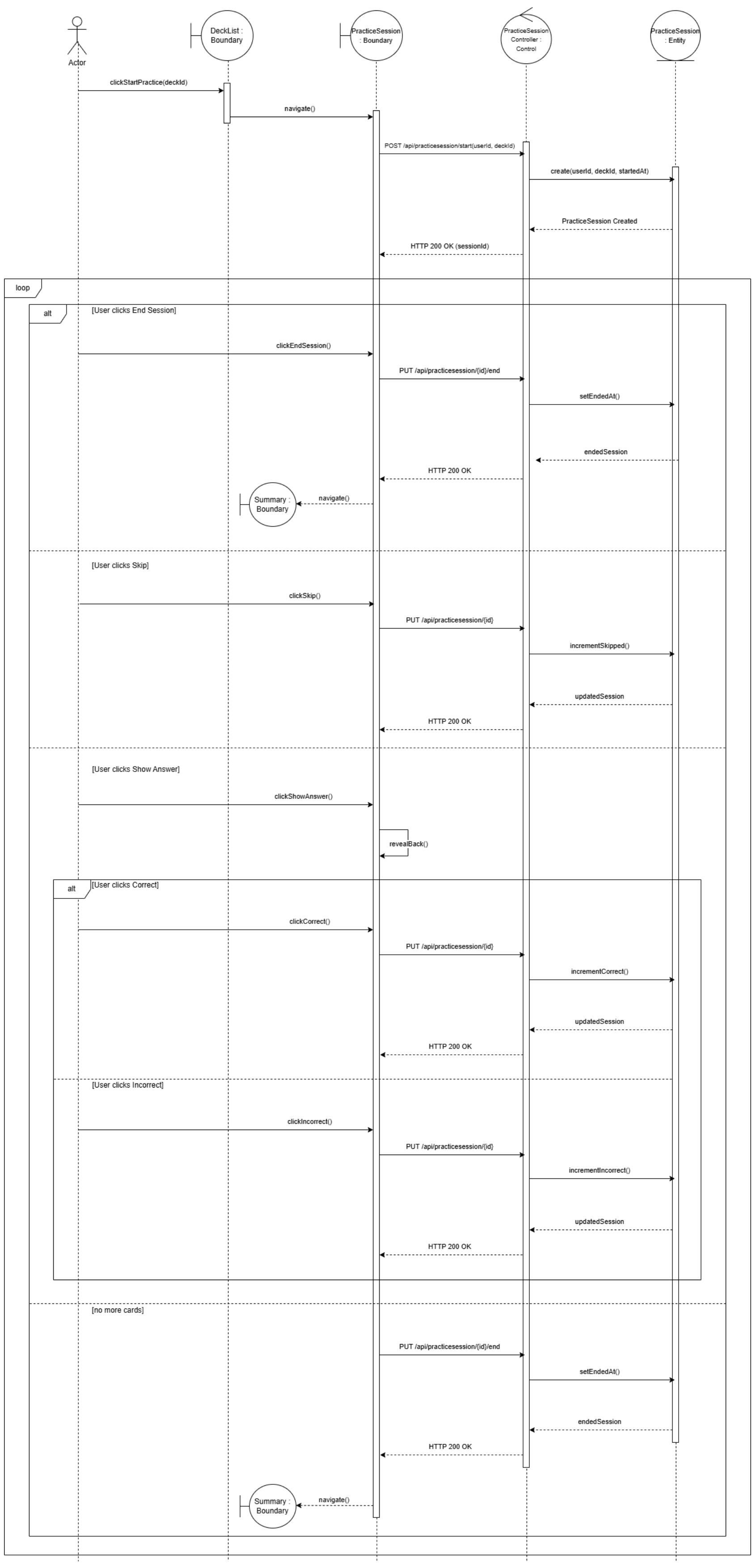


Figure 14: Practice Session Sequence Diagram

The process begins when the user selects a deck from the deck list and starts a practice session. An HTTP POST request is sent to the practice session controller, which verifies the existence of the user and the deck. If no active session exists, a new practice session is created and its identifier is returned to the interface. The system then retrieves the cards of the selected deck and displays them sequentially to the user. For each displayed card, the user may perform one of several actions: skip the card, reveal the answer and mark it as correct or incorrect, or end the session at any time. Each action sends an HTTP request to the practice session controller, which updates the session counters accordingly and returns an HTTP response confirming the update. This interaction is repeated for each card until the session ends. When the user ends the session manually or when all cards have been processed, the controller records the session's end time and returns a success response. The interface then redirects the user to the session summary page, where the final results of the practice session are displayed.

Conclusion

This chapter presented the conceptual study of the MemoryDeck application. First, the adopted UML-based design methodology was introduced. Then, use case diagrams were used to describe the main interactions between the user and the system. The class diagram defined the structural organization of the system and the relationships between its core entities. Finally, sequence diagrams illustrated the dynamic behavior of the system and the execution flow of its principal functionalities. By modeling the system through these UML diagrams, the behavior, interactions, and structure of MemoryDeck were clearly defined prior to implementation. This conceptual foundation facilitates the understanding of the system and prepares the transition to the realization phase, which is presented in the next chapter.

Chapter 3: Realization

Chapter 3: Realization

Introduction

In this last chapter, we focus on the realization phase, which represents the final step of the software development process. This stage corresponds to the concrete implementation of the MemoryDeck system, where the conceptual models defined in the previous chapter are translated into a functional web application. We begin by presenting the overall system architecture and the adopted architectural approach, followed by the technologies and development tools used during implementation. Finally, we illustrate the realized solution through its main user interfaces, highlighting the core features of the application.

3.1. System Architecture

Before detailing the technologies used, it is important to present the global architecture adopted for MemoryDeck. The application is designed with a clear separation of concerns between user interface, application logic, and data management.

3.1.1. Three-Tier Architecture

MemoryDeck is implemented using a layered web architecture that separates the system into three main layers:

- **Presentation Layer**: This layer corresponds to the user interface of the application. It is implemented using Angular and is responsible for displaying data, handling user interactions, and communicating with the backend through HTTP requests.

- **Application and Business Logic Layer**: Implemented using ASP.NET Core, this layer contains the core logic of the application. It processes user requests, applies business rules, and coordinates communication between the presentation layer and the data access layer.

- **Data Access Layer**: This layer is responsible for data persistence and database operations. It is implemented using Entity Framework Core, which manages entities, performs CRUD operations, and ensures data consistency through an object-relational mapping mechanism.

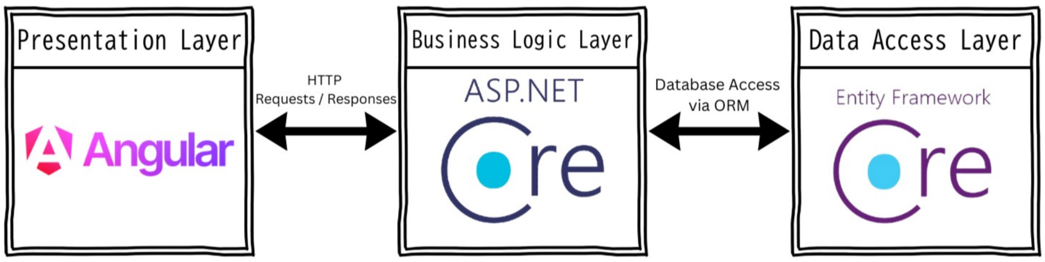


Figure 15: Three-Tier Architecture

3.1.2. Frontend–Backend Interaction

The frontend is developed using Angular and follows a component-based architecture. The user interface is structured into independent and reusable components, each responsible for its own presentation and interaction logic. These components communicate with backend services to request or update data. The backend is implemented as a controller-based Web API using ASP.NET Core. Controllers expose RESTful endpoints that receive HTTP requests from the frontend, process them using application logic, and return structured responses. Data persistence is handled transparently through Entity Framework Core. Communication between the frontend and backend is achieved through HTTP requests and responses, while interaction with the database is performed through an ORM-based access layer.

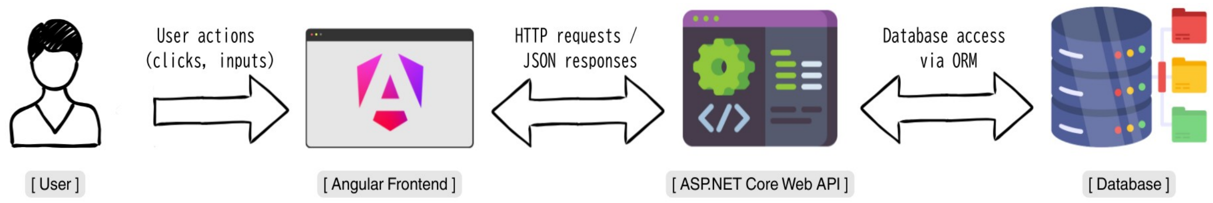


Figure 16: Frontend–Backend Interaction

3.2. Technologies and Development Tools

The development of the MemoryDeck application relied on a set of technologies, tools, and programming languages that together supported the implementation, testing, and documentation of the system.

3.2.1. Development Environment and Tools

The development of the MemoryDeck application required several software tools to support coding, testing, and documentation activities:

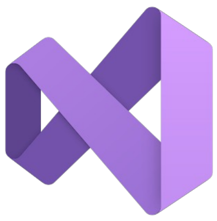


Figure 17: Visual Studio 2022

Visual Studio 2022 is a full-featured integrated development environment (IDE) developed by Microsoft. It is designed to support the development of large-scale applications using the .NET platform [6]. In this project, Visual Studio 2022 was used to develop and run the ASP.NET Core backend Web API, with integrated support for tools such as Swagger to facilitate API testing and documentation.

Figure 18: Visual Studio Code

Visual Studio Code is a lightweight and cross-platform source code editor developed by Microsoft. It supports a wide range of programming languages through extensions and offers features such as syntax highlighting, integrated terminal, and code debugging [7]. In this project, Visual Studio Code was used to develop the Angular-based frontend of MemoryDeck.

Figure 19: Swagger

Swagger is an open-source framework used to document and test RESTful APIs. When integrated with ASP.NET Core, it provides an interactive user interface that displays available API endpoints, their parameters, and response formats. This allows developers to test backend services directly from the browser without requiring an external client [8]. In this project, Swagger was enabled during development to verify API behavior and ensure correct communication between the frontend and backend.

Figure 20: Draw.io

Draw.io is a web-based diagramming tool used to create visual representations such as flowcharts and UML diagrams. It provides a wide range of shapes and modeling elements, making it suitable for software design documentation [9]. In this project, Draw.io was used to design the UML diagrams presented in the conceptual study, including use case, class, and sequence diagrams.

3.2.2. Frameworks and Technologies

Figure 21: Angular

Angular is a frontend framework used to build dynamic and structured web applications. It provides a component-based architecture, routing mechanisms, and tools for managing application state and user interactions [10]. In MemoryDeck, Angular was used to implement all user interfaces and client-side logic.

Figure 22: ASP.NET Core

ASP.NET Core is a backend framework used to develop high-performance and scalable web applications and APIs. It supports RESTful service development and request handling [11]. In this project, ASP.NET Core was used to implement the backend Web API and application logic.

Entity Framework Core is an Object-Relational Mapping (ORM) framework used to manage database interactions. It simplifies database access by allowing developers to manipulate data using object-oriented concepts [12]. It was used in MemoryDeck to manage entities, perform database operations, and ensure data consistency without writing raw SQL queries.

3.2.3. Programming Languages

Several programming languages were used in the development of the MemoryDeck application, each serving a specific role within the system.

Figure 23: HTML

HTML (HyperText Markup Language) is a standard markup language used to define the structure and content of web pages. It allows developers to organize elements such as forms, buttons, and text within a web interface. In MemoryDeck, HTML was used within Angular templates to structure the application's user interfaces.

Figure 24: CSS

CSS (Cascading Style Sheets) is a styling language used to control the visual presentation of web pages, including layout, colors, spacing, and typography. In this project, CSS was used to style the Angular components and define the visual appearance of the application. MemoryDeck focuses prioritizes desktop usability, with responsive design identified as a future enhancement.

Figure 25: TypeScript

TypeScript is a strongly typed superset of JavaScript that adds static typing and object-oriented features. It improves code readability, maintainability, and error detection during development. In MemoryDeck, TypeScript was used to implement the frontend logic within Angular components and services, handling user interactions, data binding, and communication with the backend API.

Figure 26: C#

C# is an object-oriented programming language developed by Microsoft and widely used within the within the .NET framework. It supports strong typing, modularity, and robust application development. In MemoryDeck, C# was used to implement the backend logic using ASP.NET Core, including controllers, data models, and application services, as well as integration with Entity Framework Core for database operations.

3.3. Presentation of Interfaces

This section presents the main user interfaces of the MemoryDeck web application, illustrating the core functionalities available to the user.

3.3.1. Registration Interface

The registration interface allows new users to create an account in the MemoryDeck application. Users are required to provide the necessary information to register and gain access to the system. Once the registration process is completed successfully, the user can proceed to the login interface to authenticate and start using the application's features.

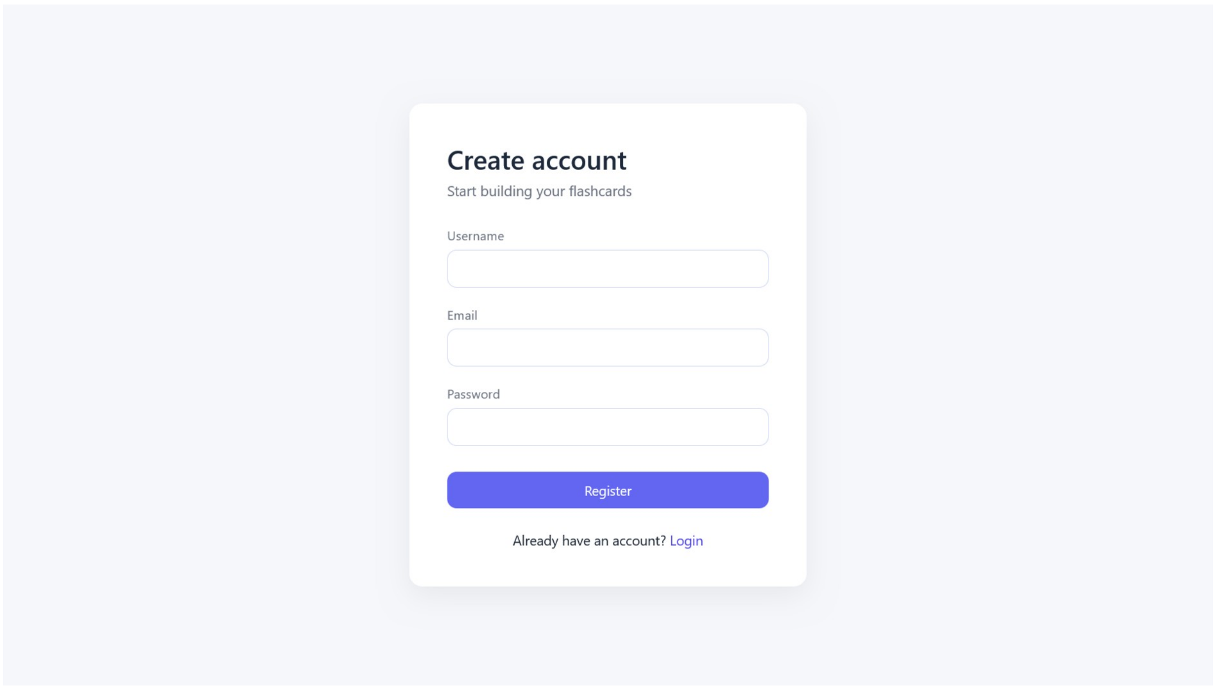

Create account
Start building your flashcards
Username
Email
Password
Register
Already have an account? Login

Figure 27: Registration Interface

3.3.2. Login Interface

The login interface allows registered users to authenticate by entering their credentials. After successful authentication, the user session is maintained using local storage, enabling access to protected features of the application without requiring repeated authentication during the same session.

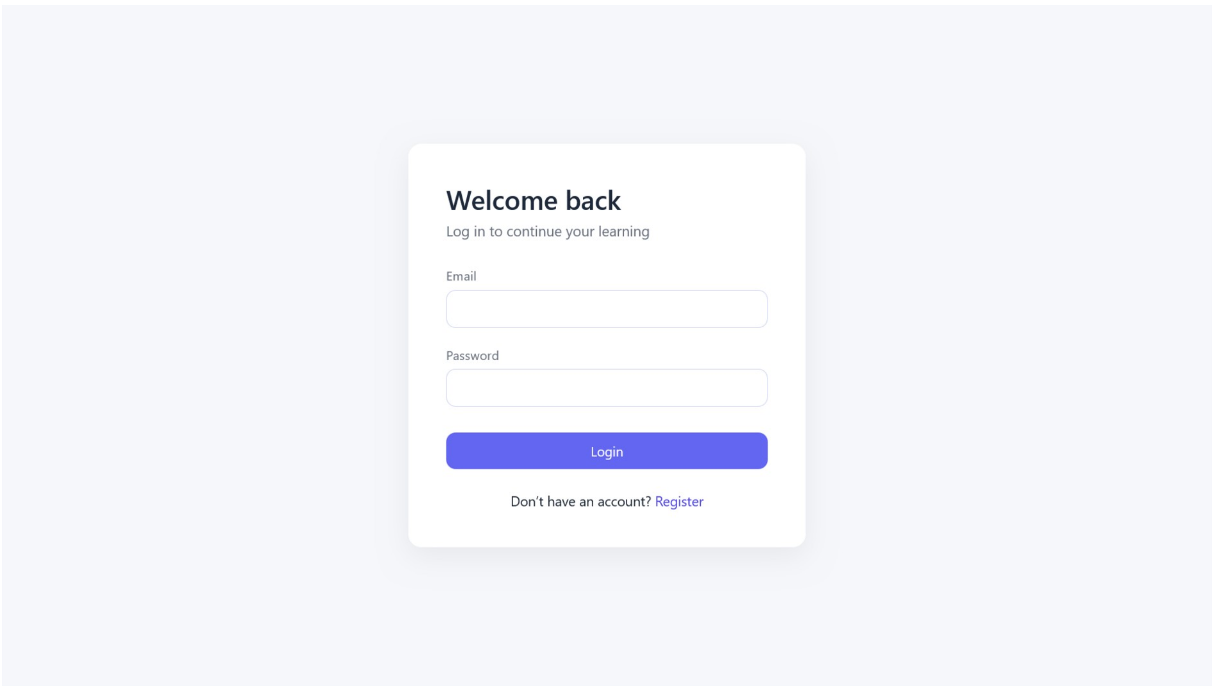

Welcome back
Log in to continue your learning
Email
Password
Login
Don't have an account? Register

Figure 28: Login Interface

3.3.3. Deck List Interface

The deck list interface displays all decks created by the user. From this page, the user can create new decks, view the details of existing ones, or delete them. Each deck entry also includes a practice button, allowing the user to directly start a practice session for the selected deck. This interface serves as the main entry point for deck management and practice initiation.

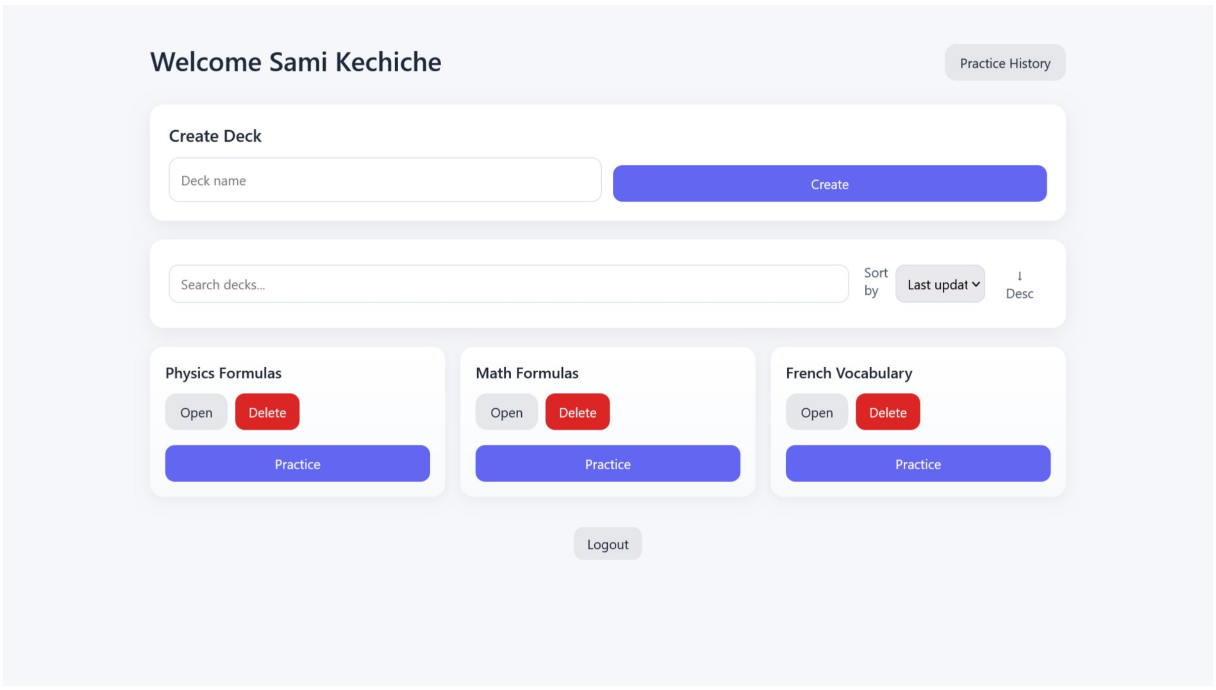
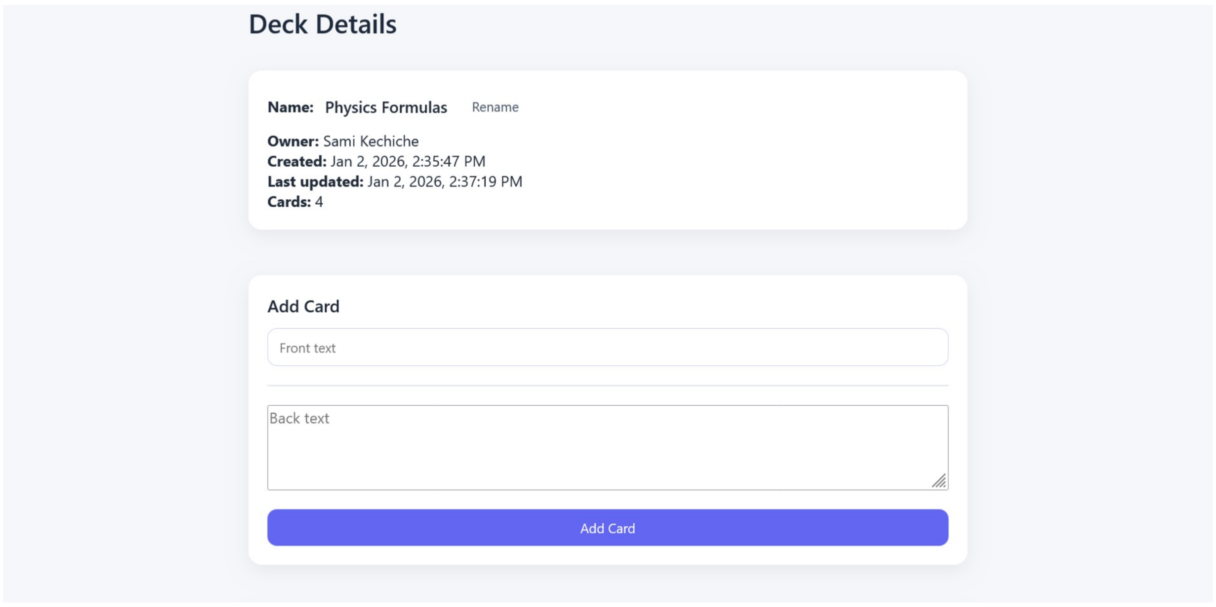


Figure 29: Deck List Interface

3.3.4. Deck Details Interface

The deck details interface provides a detailed view of a selected deck. It displays general information such as the deck name, owner, creation date, last update date, and the number of cards it contains. This interface also includes a section for creating new cards, a list of existing cards, and options to edit or delete card content.



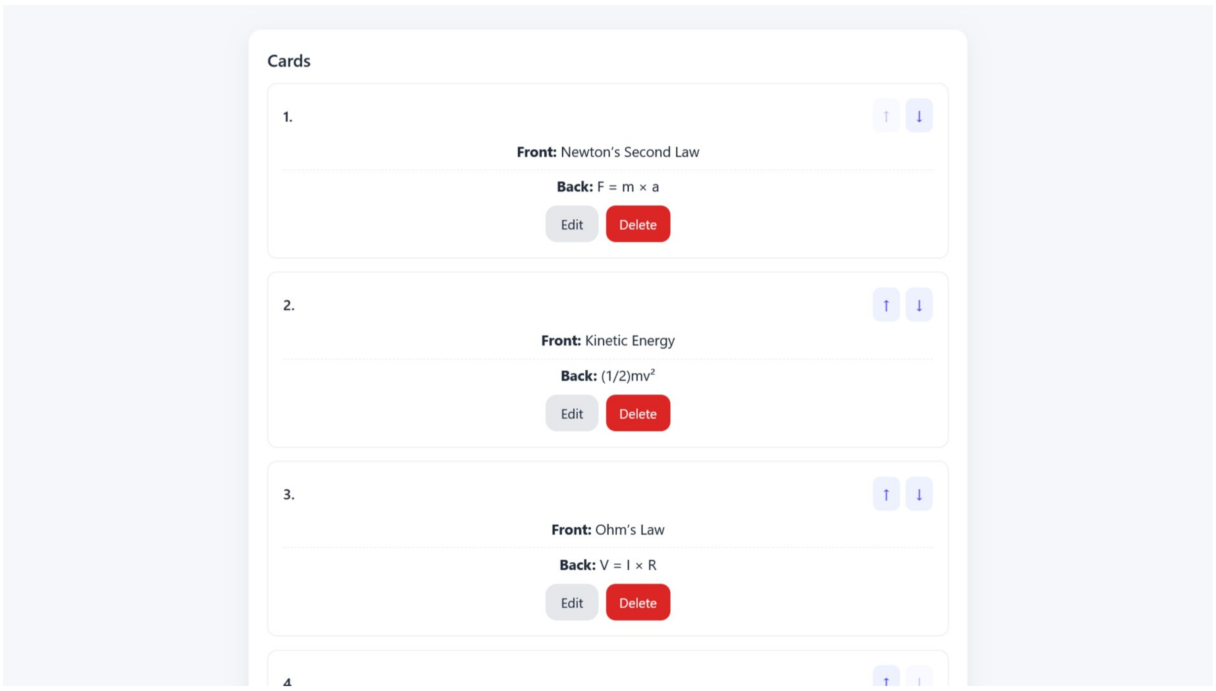
Deck Details

Name: Physics Formulas

Owner: Sami Kechiche
Created: Jan 2, 2026, 2:35:47 PM
Last updated: Jan 2, 2026, 2:37:19 PM
Cards: 4

Add Card

Figure 30: Deck Details Interface



Cards

1.
Front: Newton's Second Law
Back: F = m × a

2.
Front: Kinetic Energy
Back: (1/2)mv²

3.
Front: Ohm's Law
Back: V = I × R

4.

Figure 31: Deck Details Interface (Card List)

3.3.5. Practice Session Interface

The practice session interface guides the user through the learning process by displaying cards one at a time. During the session, the user can reveal the answer, mark the response as correct or incorrect, skip the current card, or end the session at any time.

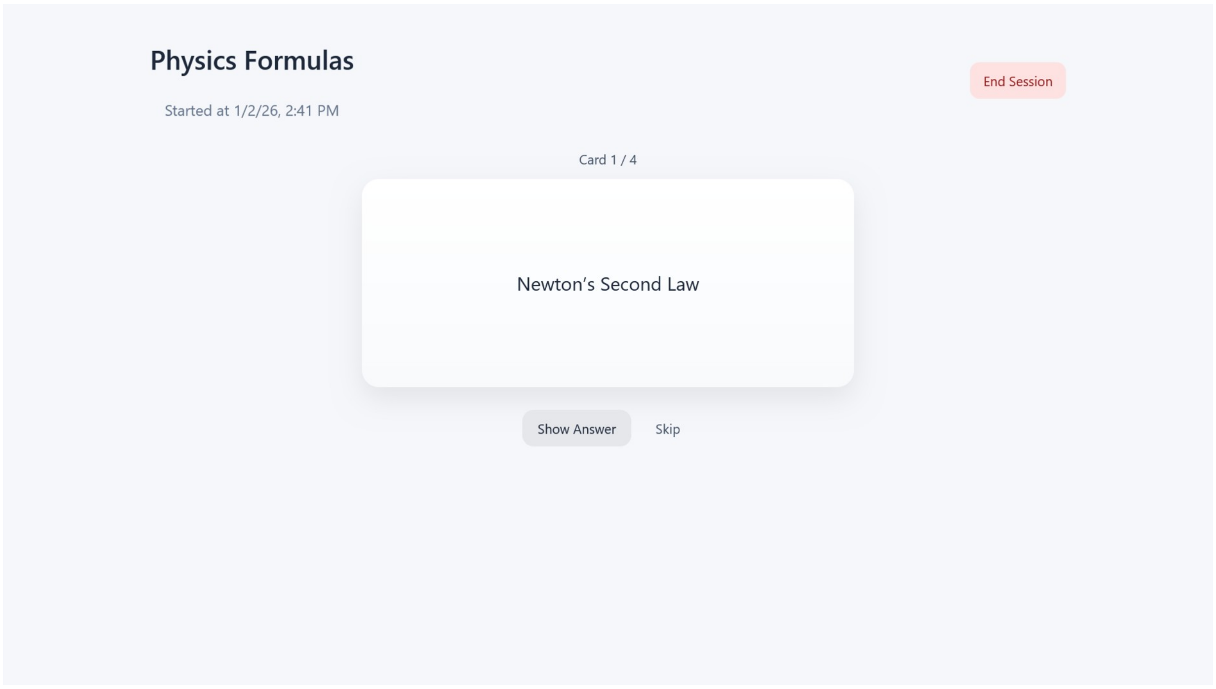


Figure 32: Practice Session Interface - Front of Card

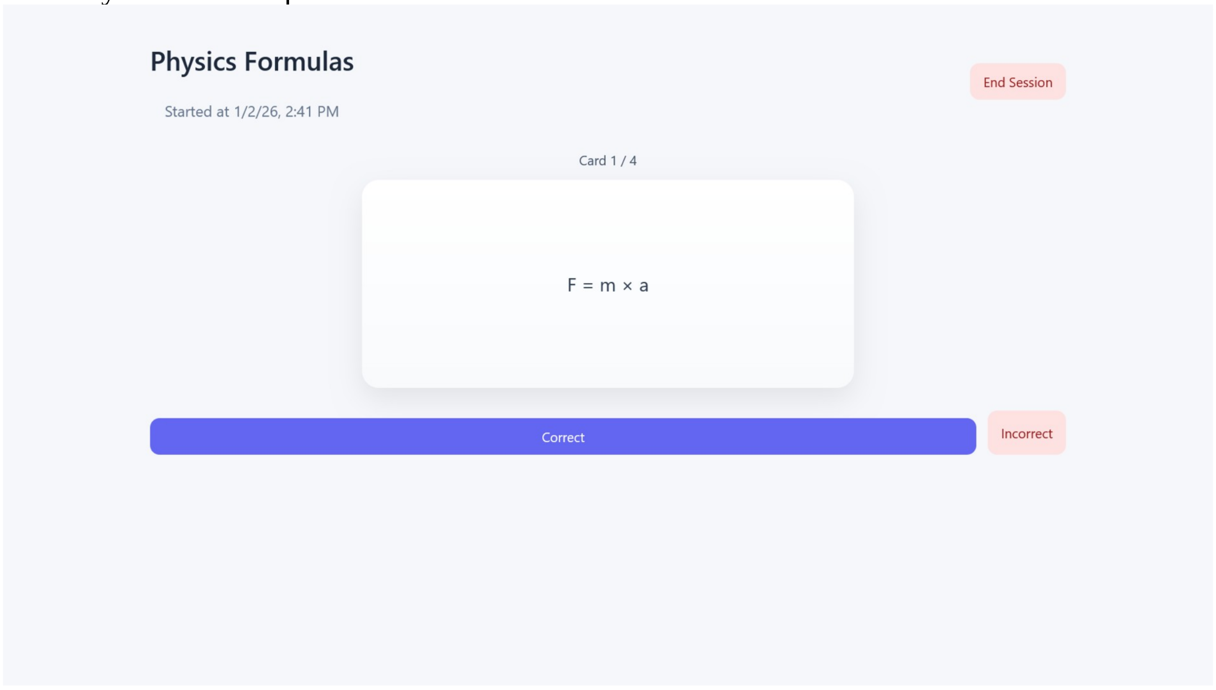
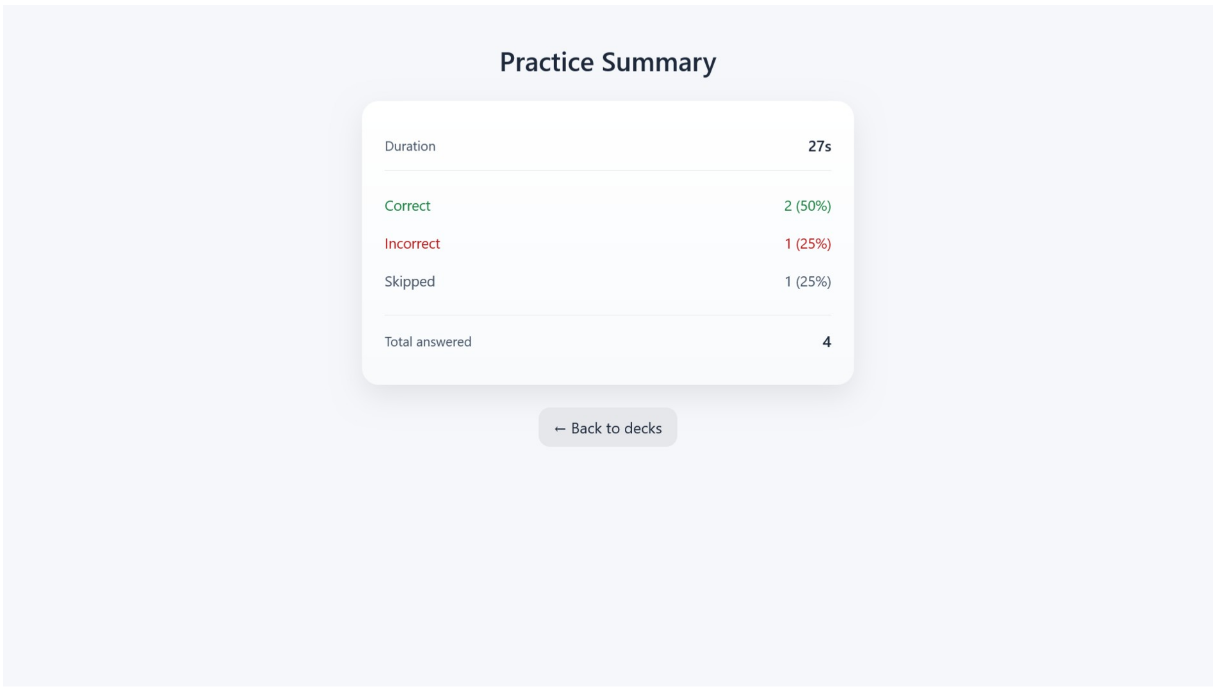


Figure 33: Practice Session Interface - Back of Card Revealed

3.3.6. Practice Session Summary

At the end of a practice session, a summary is displayed showing session results, including the number of correct, incorrect, and skipped cards.



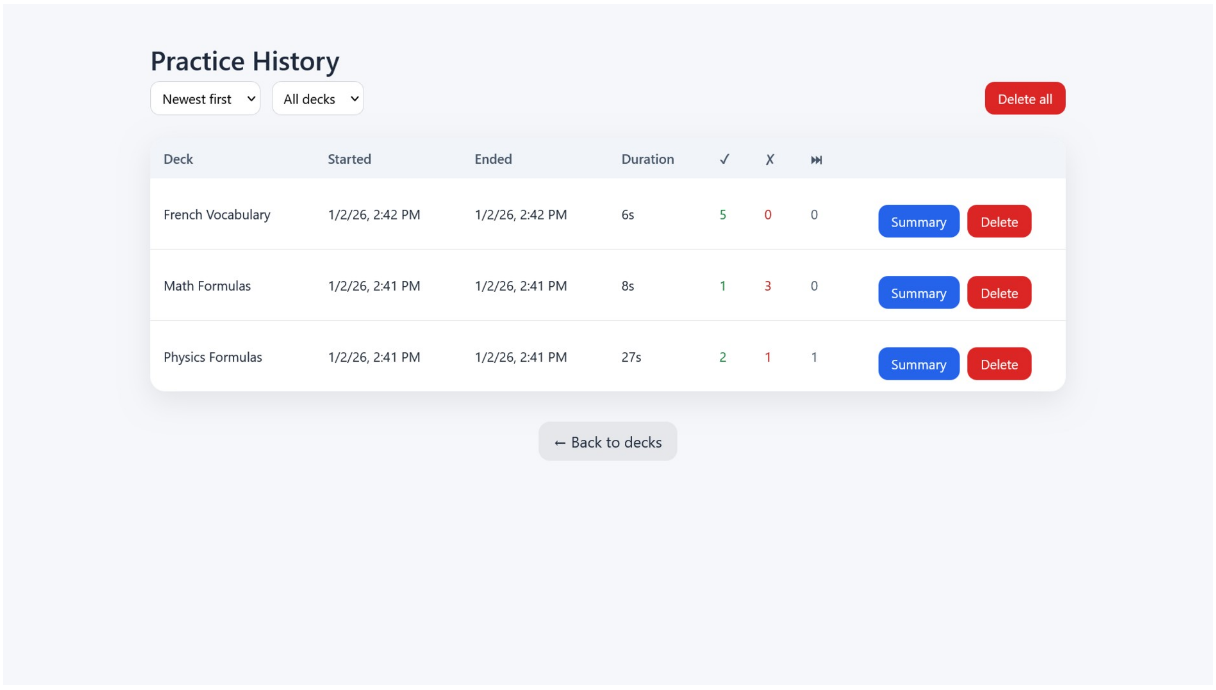
Practice Summary

Duration	27s
Correct	2 (50%)
Incorrect	1 (25%)
Skipped	1 (25%)
Total answered	4

Figure 34: Practice Session Summary

3.3.7. Practice Session History

The practice session history interface allows users to review all previously completed practice sessions. From this interface, users can delete individual sessions or clear the entire history. The history can also be filtered by deck and sorted by date, enabling users to easily analyze their learning progress over time and focus on specific decks or periods.



Deck	Started	Ended	Duration	✓	X	⏭	
French Vocabulary	1/2/26, 2:42 PM	1/2/26, 2:42 PM	6s	5	0	0	Summary Delete
Math Formulas	1/2/26, 2:41 PM	1/2/26, 2:41 PM	8s	1	3	0	Summary Delete
Physics Formulas	1/2/26, 2:41 PM	1/2/26, 2:41 PM	27s	2	1	1	Summary Delete

Figure 35: Practice Session History

Conclusion

In this chapter, we presented the realization of the MemoryDeck application by describing its system architecture, technologies, development tools, and main user interfaces. The implementation faithfully reflects the conceptual models introduced in the previous chapter and results in a functional web application designed to support effective flashcard-based learning.

General Conclusion and Perspectives

This Integration Project resulted in the design and development of **MemoryDeck**, a web-based flashcard application intended to support effective and structured learning through active recall. The project was carried out within the academic framework of the fourth year in Software Engineering at EPI Digital School and aimed to apply theoretical knowledge to a concrete and functional software solution.

The work began with a preliminary study that introduced the project context, analyzed existing learning tools, and highlighted their limitations. This analysis made it possible to define a clear set of functional and non-functional requirements adapted to the needs of learners seeking a simple and focused study tool. A detailed conceptual study followed, relying on UML modeling to describe the system's behavior, structure, and interactions through use case, class, and sequence diagrams. This phase ensured a clear understanding of the system before implementation.

The realization phase led to the development of a functional web application based on a three-tier architecture. The frontend was implemented using Angular, while the backend was developed as a RESTful Web API using ASP.NET Core and Entity Framework Core for data persistence. The application provides essential features such as user registration and authentication, deck and card management, practice sessions, performance summaries, and practice history tracking.

Although the objectives of the project were successfully achieved, several perspectives for future improvement can be considered. The application could be extended with advanced study features such as randomized or adaptive practice order based on performance, tagging systems for cards, or the ability to organize decks hierarchically. Additional functionalities could include sharing decks between users, richer practice statistics with long-term progress tracking, timers, streaks, and automatic backup mechanisms. Support for multimedia content within cards, advanced formatting options, and new card types could further enhance the learning experience. From an architectural perspective, introducing additional user roles, such as teachers, could open the door to collaborative or supervised learning scenarios.

In conclusion, MemoryDeck represents a coherent and functional learning application that meets the initial requirements and demonstrates the practical application of software engineering concepts. This project provided valuable experience in full-stack web development, system modeling, and architectural design, and constitutes a solid foundation for more advanced developments in future academic or professional projects.

Webography

[1] Host Institution Official Website https://www.epi-digital-school.tn [Consulted on: October 12, 2025]

[2] Anki – Official Website https://apps.ankiweb.net [Consulted on: October 20, 2025]

[3] Quizlet – Official Website https://quizlet.com [Consulted on: October 22, 2025]

[4] Kahoot! – Official Website https://kahoot.com [Consulted on: October 24, 2025]

[5] Unified Modeling Language (UML) – Official Specification https://www.uml.org [Consulted on: October 28, 2025]

[6] Visual Studio 2022 – Official Documentation https://visualstudio.microsoft.com [Consulted on: November 14, 2025]

[7] Visual Studio Code – Official Website https://code.visualstudio.com [Consulted on: November 16, 2025]

[8] Swagger / OpenAPI Documentation https://swagger.io/docs/ [Consulted on: December 7, 2025]

[9] Draw.io (diagrams.net) – Official Website https://www.diagrams.net [Consulted on: October 30, 2025]

[10] Angular – Official Documentation https://angular.io/docs [Consulted on: November 6, 2025]

[11] ASP.NET Core – Official Documentation https://learn.microsoft.com/aspnet/core [Consulted on: November 18, 2025]

[12] Entity Framework Core – Official Documentation https://learn.microsoft.com/ef/core [Consulted on: November 25, 2025]