Assignment 3

Team number: <Your group number on Canvas>
Team members

Name	**Student Nr.**	**Email**
Yitong Tang	2798775	yttangfdu@gmail.com
Jing Chen	2801366	jingc4853@gmail.com
Menghan Zhang	2807975	menghan.zhang2023@gmail.com
Kota Sree Pragnya	2788696	pragnyaarj@gmail.com

Summary of changes from Assignment 2

Author(s): name of the team member(s) responsible for this section

Provide a bullet list summarising all the changes you performed in Assignment 2 for addressing our feedback.

Maximum number of pages for this section: 1

Class Diagram

1. The addition of new classes and architecture changes between flashcards and levels necessitated updates to the class diagram.
2. The deletion of redundant classes led to modifications in the class diagram.
3. Method changes related to the transition from static to dynamic requirements influenced adjustments in the class diagram.

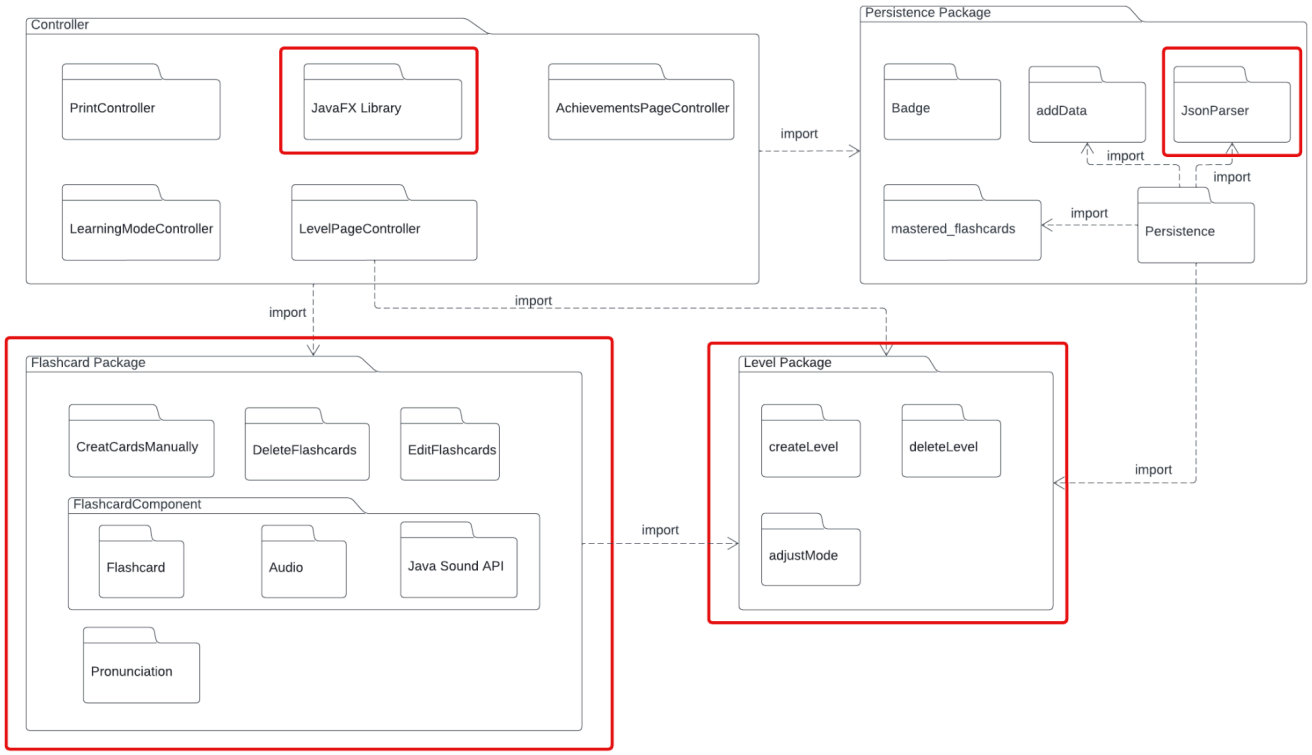
From the peer feedback for the state machine diagram the suggested changes have been implemented in both diagrams to enhance their clarity, completeness, and readability.

1. In both the state machine diagrams, guards have been added to all necessary transitions, ensuring that state transitions occur under appropriate conditions.
2. Additionally, a submachine state has been employed for the "Learning Mode" state to better encapsulate its complex structure, improving overall clarity and readability. Image quality issues have been addressed, ensuring that all labels are now readable and size is also clear.
3. In the state machine diagram "Learning mode," labels have been added for all transitions and also == which was missed in our previous diagram is now modified.

These modifications have significantly improved both diagrams, making them more effective communication tools for understanding system behaviour.

Revised package diagram

Author(s): Yitong & Jing & Sree & Menghan



The package diagram is mainly revised in three parts, they are external libraries, flashcard package and level package.

1.External library

We add external libraries in the package diagram, as they are also imported in the system. According to the class diagram, we can see that the MainController uses the **J**avaFX Library, so the **Controller package** should include the **JavaFX Library**. As we use JsonParser while exporting the json file, the **Persistence Package** should import the **JsonParser package**. The Flashcard uses Java Sound API, so **Java Sound API** is part of the **FlashcardComponent package**.

2.Flashcard Package

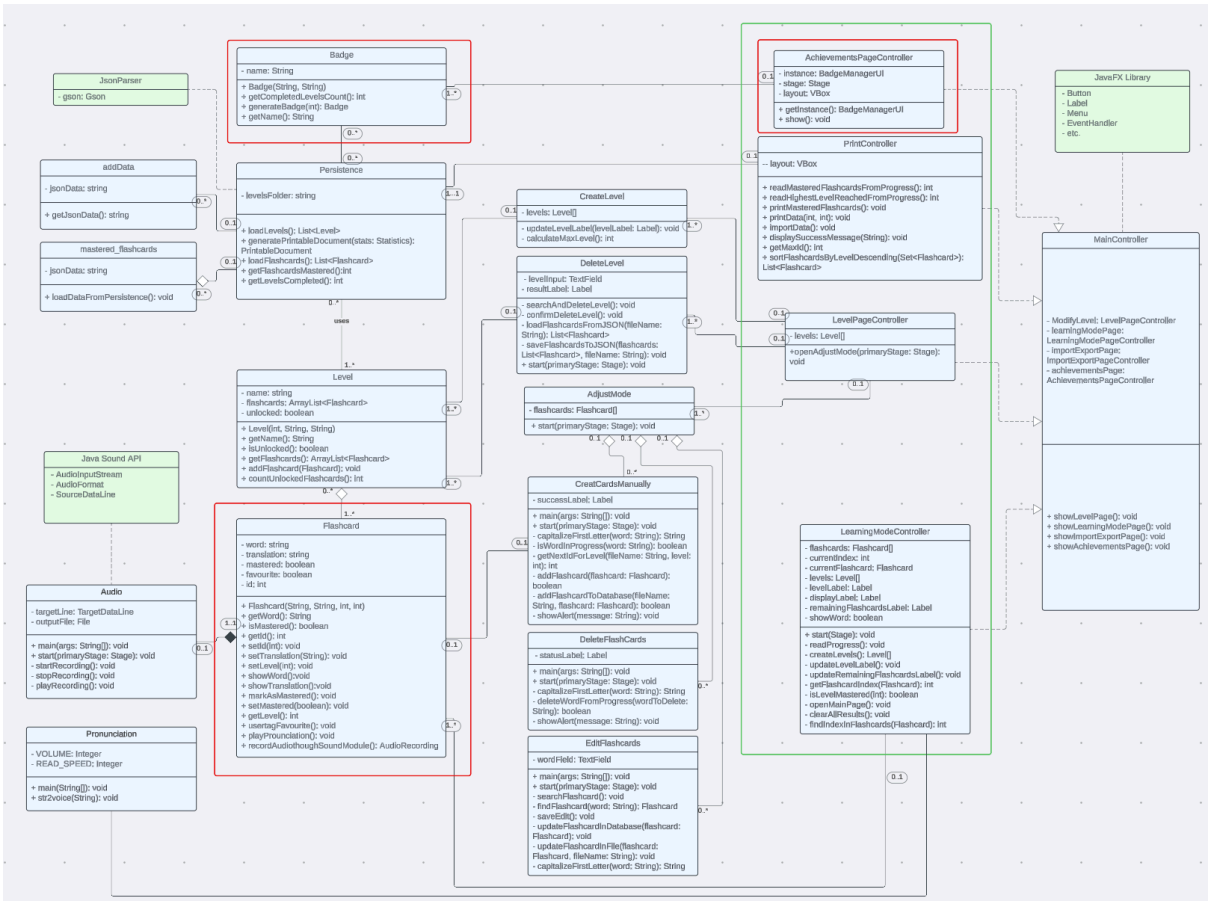
As the Flashcard now has several subclasses, the **Flashcard package** includes three subpackages, which are **CreateCardsManually package**, **DeleteFlashcards package** and **Edit Flashcards package**. It also has the **FlashcardComponent package** and **Pronunciation package** for the audio and pronunciation part.

3.Level

The Level class consists of three subclasses, so the subpackages, **createLevel package**, **deleteLevel package** and **adjustMode package** should be included in the **Level package**.

Revised class diagram

Author(s): Yitong



Design Method:

1.In JavaFX, the concept of Inversion of Control is realised through the use of FXML files and Controller classes. We segment the system into various functional modes. Considering the GUI aspect, we incorporated controllers to facilitate page presentation. Consequently, the class diagram is divided into five parts, presented from right to left.

a. Detailed Functions Linked with Main Class:

These classes, such as audio, and addData, are closely associated with essential classes like flashcards and persistence. They either constitute a part of them or have a tight relationship. Typically, they interface with external libraries.

b. Important Classes:

This segment comprises vital components of the system: badge, persistence, level, and flashcard. These classes possess their attributes and operate on the previous tier's classes. For example, the flashcard class has properties like word, translation, and mastered status, along with functions such as showWord(), which is related to its own attribute, and it also has interaction with the SoundModule like playPronunciation().

c. Controller Model Classes:

These classes represent different modes accessible to the user, primarily for managing essential classes and connecting with user information. Taking flashcards as an example, in

the LearningModeController, users can add, remove, or display flashcards while recording user-related activities like saveResults().

d. Main Controller:

These classes form the overall control category. Controllers serve as intermediaries during mode transitions, facilitating user interactions with the system.

2. Singleton design pattern

In our design, many classes' data are subject to change. To maintain higher consistency, we attempted to use the singleton pattern for more classes, ensuring only one instance exists in the application. **AchievementsPageController** is designed as a singleton, and some other classes, such as Persistence, can also be designed as singletons.

3. Object identity, equality, uniqueness

In the initial design, the **Flashcard class** lacked a unique identifier, relying instead on a combination of level and ID. However, this approach posed challenges, including issues with maintaining word order within levels and complexity in ID assignment. Consequently, a unique ID was adopted based on the order of entry into the database, ensuring clarity and eliminating concerns about duplicate IDs or confusion. Integrating unique identifiers enhances object identification, comparison accuracy, and data integrity, simplifying system logic and operations. This adjustment aligns with applied design patterns, promoting clarity, reliability, and efficiency in Flashcard management.

4. Factory method

In designing our **Badge class**, we aimed to provide a seamless way to represent user achievement levels while shielding client code from the complexity of badge creation. We opted for the Factory Method pattern, encapsulating badge creation in a single method, generateBadge(int totalLevelsCompleted). This method dynamically determines the appropriate badge type based on the user's completed levels. By abstracting the creation process, we ensure that client code remains unchanged regardless of the badge type. This approach enhances code modularity, simplifies maintenance, and promotes scalability as our badge system evolves.

5.Other Change1: Architecture Changes Between Flashcards and Levels:

In our previous design iteration, while we correctly applied the composition relationship between flashcards and levels, we overlooked their interrelation. During the implementation phase, I handled the CRUD operations of flashcards and levels in parallel. However, I realised that the adjustment of flashcards should be encapsulated within the adjustment of levels. Upon this realisation, we restructured the LevelPageController and FlashcardPageController classes. We redesigned flashcards as components of the adjustment mode of levels. This change simplifies and clarifies the process of updating and saving data.

6.Other Change2: Deletion of Redundant Classes:

In our previous design, we identified some redundant classes, such as `HomePage` and AudioRecording. In this version, we merged certain classes to enhance the overall cleanliness and organisation of the design.

7.Other Change3: Transition from Static to Dynamic Requirements:

During the initial class design phase, our focus was primarily on structural aspects. While class design inherently involves structural considerations, upon comparing our class design with actual code implementation, we discovered numerous operations that we had overlooked. These omitted functionalities typically relate to the transmission and exchange of information between classes. Therefore, in our reevaluation, we added more features

related to information exchange, as well as getter and setter methods, to better accommodate dynamic requirements.

Application of design patterns

Author(s): Menghan

	DP1
Design pattern	Object identity, equality, uniqueness
Problem	We need to ensure the uniqueness of each flashcard object to track the user's learning progress on each card. The system should identify flashcard objects accurately when users are learning.
Solution	We ensure uniqueness by assigning a unique id to each Flashcard object.
Intended use	The system uses the unique id to identify and distinguish each Flashcard. The id is assigned at object creation and ensures that even if two Flashcard objects have the same word and translation, they are considered different objects due to their distinct id.
Constraints	If the uniqueness strategy for Flashcards needs to be adjusted in the future, it could involve a lot of code refactoring.

	DP2
Design pattern	Inversion of Control
Problem	In traditional control flow, there will be limitations like high coupling, low extensibility and high complexity.
Solution	In JavaFX, we use FXML files and the Controller class to implement Inversion of Control . The FXML file is responsible for defining the UI layout, while the Controller class is responsible for handling user interaction and business logic.
Intended use	The JavaFX framework is responsible for handling user input events (such as button clicks) and invoking the corresponding event handlers. For example, when the user clicks on the "Learning Mode" button, the system calls the openFlashcardStudyMode method, which defines how to respond to the event rather than the program code directly controlling the flow.
Constraints	It adds complexity to the framework.

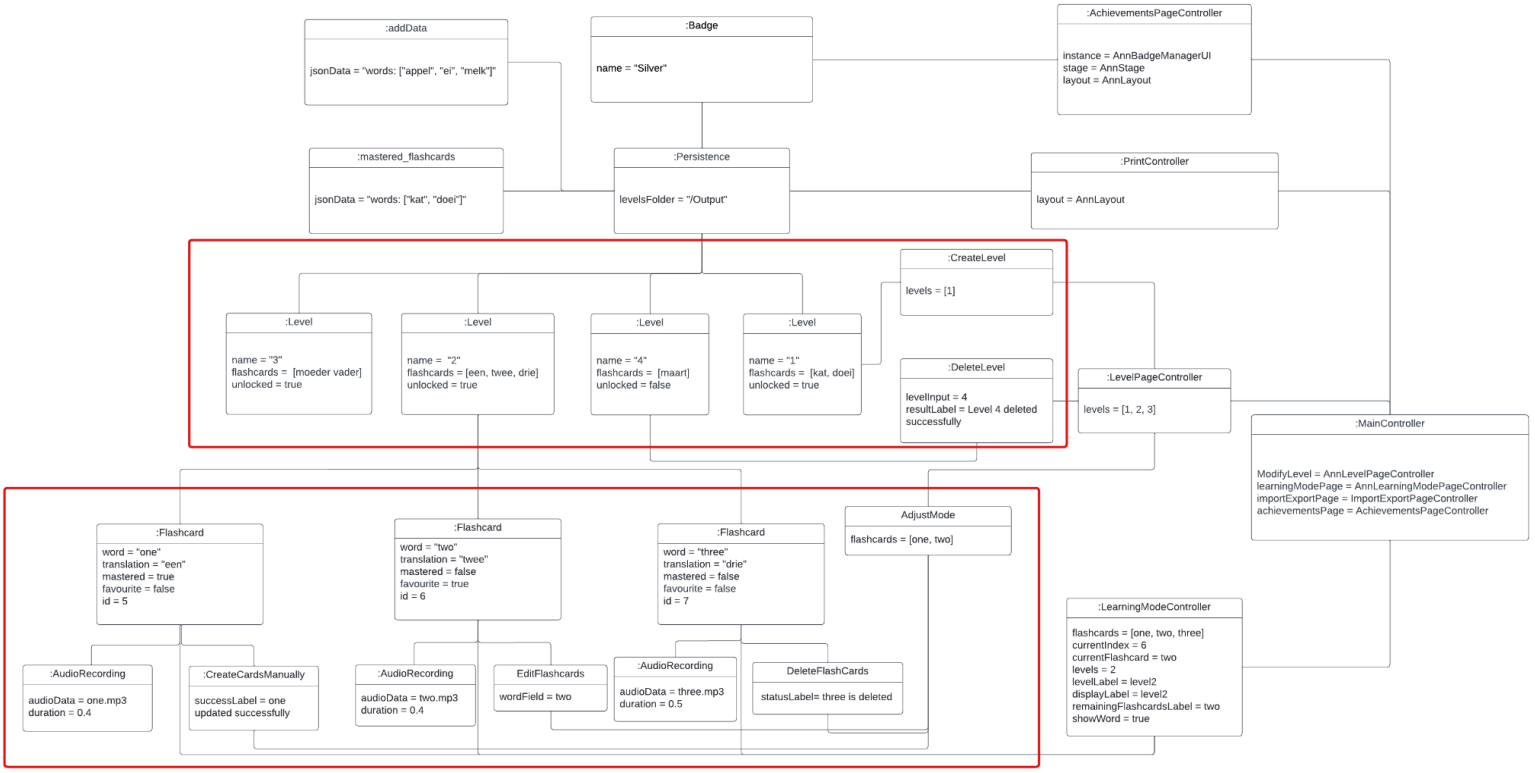
	DP3

Design pattern	Singleton
Problem	We need a concentrated way to manage and display the badges earned by users. We should avoid multiple instances of the management interface to ensure the consistency of the UI and synchronisation of the state.
Solution	We ensure that there is only one instance of BadgeManagerUI accessible from any point in the program. When the user completes a level, we can update the badge status in real-time and maintain UI consistency.
Intended use	We use a public static method getInstance(). It first checks whether the instance has been created. If not, BadgeManagerUI creates a new instance and assigns it to the instance; if it already exists, it returns the instance directly. This ensures that no matter how many times getInstance() is called, there will only be one instance of BadgeManagerUI during the entire program's runtime.
Constraints	If not properly implemented with thread safety in mind, it may result in multiple instances in a multithreaded environment.

	DP4
Design pattern	Factory method
Problem	We need to create badge objects that represent different levels of user achievement without exposing the creation logic to the client code. As users complete different levels, they should receive badges accordingly, but the client code should remain unchanged regardless of the type of badge created.
Solution	We encapsulate the object creation process of badges in a single method, generateBadge(int totalLevelsCompleted). It determines the type of badge to instantiate based on the number of levels completed by the user. The client code simply calls this method and relies on it to return the appropriate badge object, thus separating the object creation responsibility from the client.
Intended use	When the system needs to award a badge to a user, it calls the generateBadge method with the user’s total completed levels as the argument. The method then uses the argument to determine which badge to create. For example, if the total completed levels are three or more, it returns a "Platinum" badge.
Constraints	It assumes that the badges are differentiated solely based on the number of levels completed. If future requirements include different criteria for badge allocation, the generateBadge method would need to be modified or extended to accommodate these changes.

[optional] Revised object diagram

Author(s): Menghan



[object diagram]

The object diagram is mainly revised in two parts, the Flashcard and the Level.

1.Level

We divide the LevelPageController class into three separate classes to make the operations more clear. Two objects link to the LevelPageController, they are CreateLevel object and DeleteLevel object. There is another object about levels called AdjustMode, which is modified through the Flashcard object. All the three objects link to LevelPageController. There are four levels in this scenario. For the **CreateLevel object**, the user created level1, so levels = [1]. For the **DeleteLevel object**, the user deleted level4, then it shows a message “Level 4 deleted successfully”. The user is learning level2 right now, and she doesn’t change level3, so there are three remaining levels. Therefore, the **LevelPageController object**, levels = [1, 2, 3].

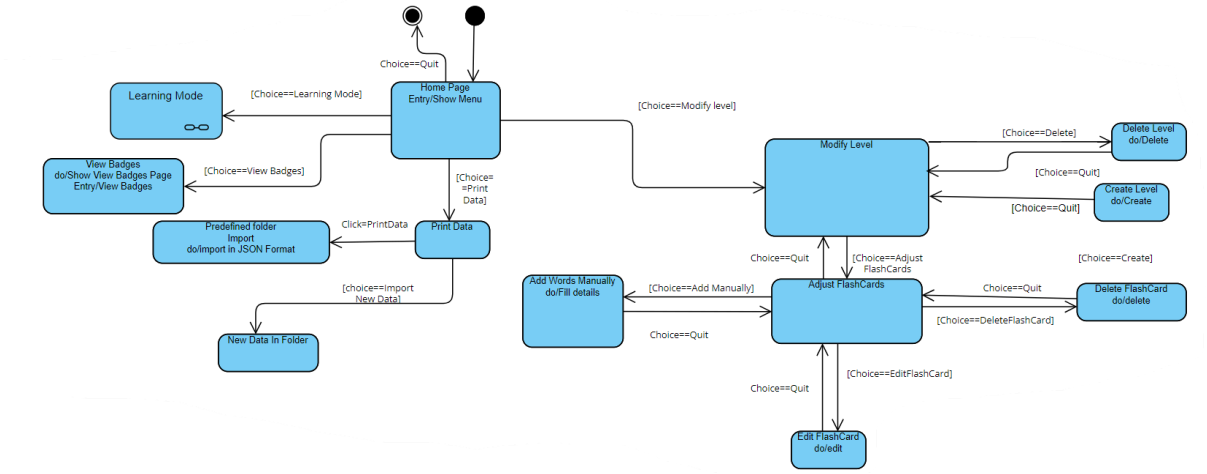
2.Flashcard

We add more flashcards in this scenario to include more possible actions. In level 2, there are three flashcards currently. For the **CreateCardsManually object**, we add “one” successfully, so it shows a message “one updated successfully”. For the **EditFlashcards object**, we change favourite = true successfully, so worldField = two. For the **DeleteFlashcards object**, we search for the word “three”, and then delete it. After that, it shows a message ”three is deleted successfully”. The words link to the **AdjustMode object**, flashcard = [one, two].

Revised state machine diagrams

Author(s): Kota Sree Pragnya

Class Name: Controller



Upon launching the application, users are greeted with the Home Page Menu, which serves as the initial state. This menu acts as a gateway to various features and functionalities within the application. When we enter the Home Page, an introduction to the application's functions and features is displayed. After the introduction, the user is seamlessly transitioned to the ***Menu Displayed*** state, providing access to all available features and when the user chooses to quit from this state they can choose to quit and close the application. We have chosen this because it gives a user-friendly introduction and creates a welcoming experience, which will help users understand the purpose and functionality of the application and will also attract more users which is definitely a benefit.

Moving to the Modify level, when users enter they will encounter several states. They can then seamlessly transition to the Adding Level, Editing Level, and Deleting Level states, depending on their actions to manage levels.They can also quit and go back to the home page if they wish.We are designing this way because centralising the level management will simplify the user interactions, and will offer a clear hierarchy for organising and enhancing user efficiency in customising their learning experience.

The Adjust FlashCard Management Page introduces users to do multiple actions on the flashcards. Like Adding Flashcard, Editing Flashcard, and Deleting Flashcard states based on their interactions to manage flashcards. Allowing all flashcards to edit in the Adjust Flashcard Management Page at any level ensures easy access and clear management. Allowing users to edit supports personalised learning experiences, accommodating diverse study preferences and improving overall usability which is what users expect in the learning mode.

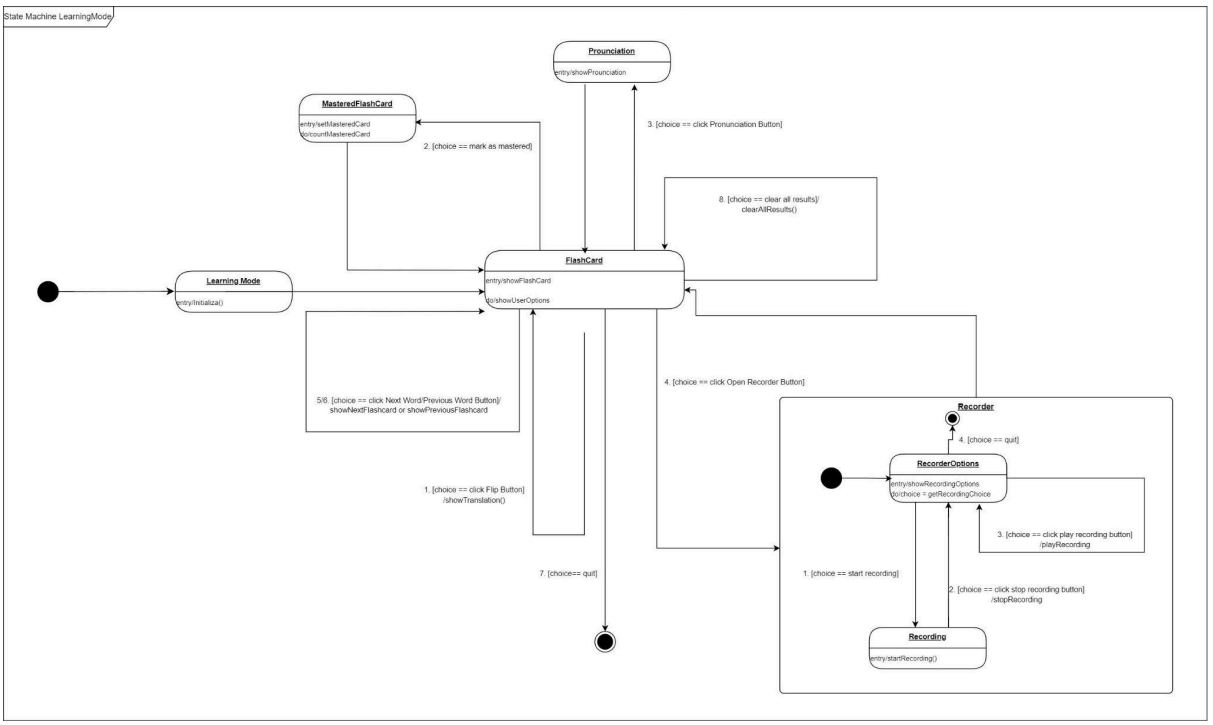
The Learning Mode Page presents the ***Flashcard Displayed*** state, where users engage with the current flashcard. We will give a detailed introduction of this part in our next state machine diagram, named “LearningMode”.

The Print Data starts when the users make a choice when they want to Print Data .When users initiate import/export operations which is to print data, they transition to the ***Print Data*** state where they click on the print data option all the information goes to a predefined folder and then continue exporting in JSON format . After successful completion, the ***Viewing Predefined Folder*** state allows users to inspect the predefined folder in which they will have the information they exported. This feature is designed to enable users to seamlessly transfer data across devices or share learning materials with others.

Lastly, the View Badges offers the ***View Badges*** state, where users can view all earned achievement badges. Accessible directly, this page provides a visual representation of user accomplishments. Displaying the earned achievements and badges on the Achievements Page serves as positive reinforcement and motivation to users to achieve learning milestones. We are making the Achievements Page directly accessible so it ensures users can readily track their progress, fostering a sense of accomplishment and engagement.

In summary, the “Controller” state machine diagram navigates users through an intuitive and user-friendly experience, allowing them to seamlessly transition between various pages and states to manage levels, flashcards, track their learning progress and view their achievements. The design acknowledges the dynamic nature of user needs and technology advancements. This forward-looking approach ensures the application remains adaptive and can incorporate enhancements based on user feedback and emerging trends.

Class Name: LearningMode



In the “LearningMode” state machine, the primary objective we have and follow is to guide users through the process of learning new words, allowing them to progress through flashcards in a step-by-step manner. The states and transitions encapsulate the key functionalities and interactions within the Learning Mode.

Firstly the learning mode starts with a state that represents the initial state when a user enters from the home page.Upon Entry the system prepares to select the wordlist and goes to the next state which is ***WordListSelection***. In the ***WorldListSelection*** state upon entry it shows all the word lists and also the first flashcard. This decision is rooted in cognitive learning principles, ensuring users grasp one concept at a time before moving forward, enhancing overall retention.

When the system enters the next state the user starts newly only level1 will be shown but later when he starts learning and finishes other levels this state also allows the users to choose the levels of their choice to review their learning .We have designed this way as to allow users to starts from basic and also they can get personalised learning experiences.

When a flash card is shown it enters Flashcard state where there are several transitions. In this state it displays the flash card front where by clicking the pronunciation it enters to the ***pronunciation*** state in which the user can listen to the pronunciation of the word and by clicking the open recorder the system will enter to the ***Recorder*** state where the user can record their pronunciation and by clicking on the play recording the system allows the user to

listen their recorded pronunciation.This design feature is designed way to integrate pronunciation and recording features to enrich the learning experience by providing users with auditory cues and allowing them to record their pronunciation so the system accommodates diverse learning styles for different type of users.

Once they click on flip the system will enter show the translation of the word .In this state it self the user has an option to go to the next card if the choice is NextWord and if the choice is PreviousWord it goes to previous card as per the users choice and this ensures a seamless flow to the next word in the learning sequence.This decision aims to minimise the interruptions in the running of the system and also will also ensure the user engagement is maintained.

In this state when the user sees the translation of the word and feels that they are confident with the translation they can exit saying they have “mastered” the word by which the system will go into the next state.This will empower the user to self-assess their confidence in understanding a word. This feature also aligns with self-directed learning principles which will give users control over their progress and promote a sense of achievement.

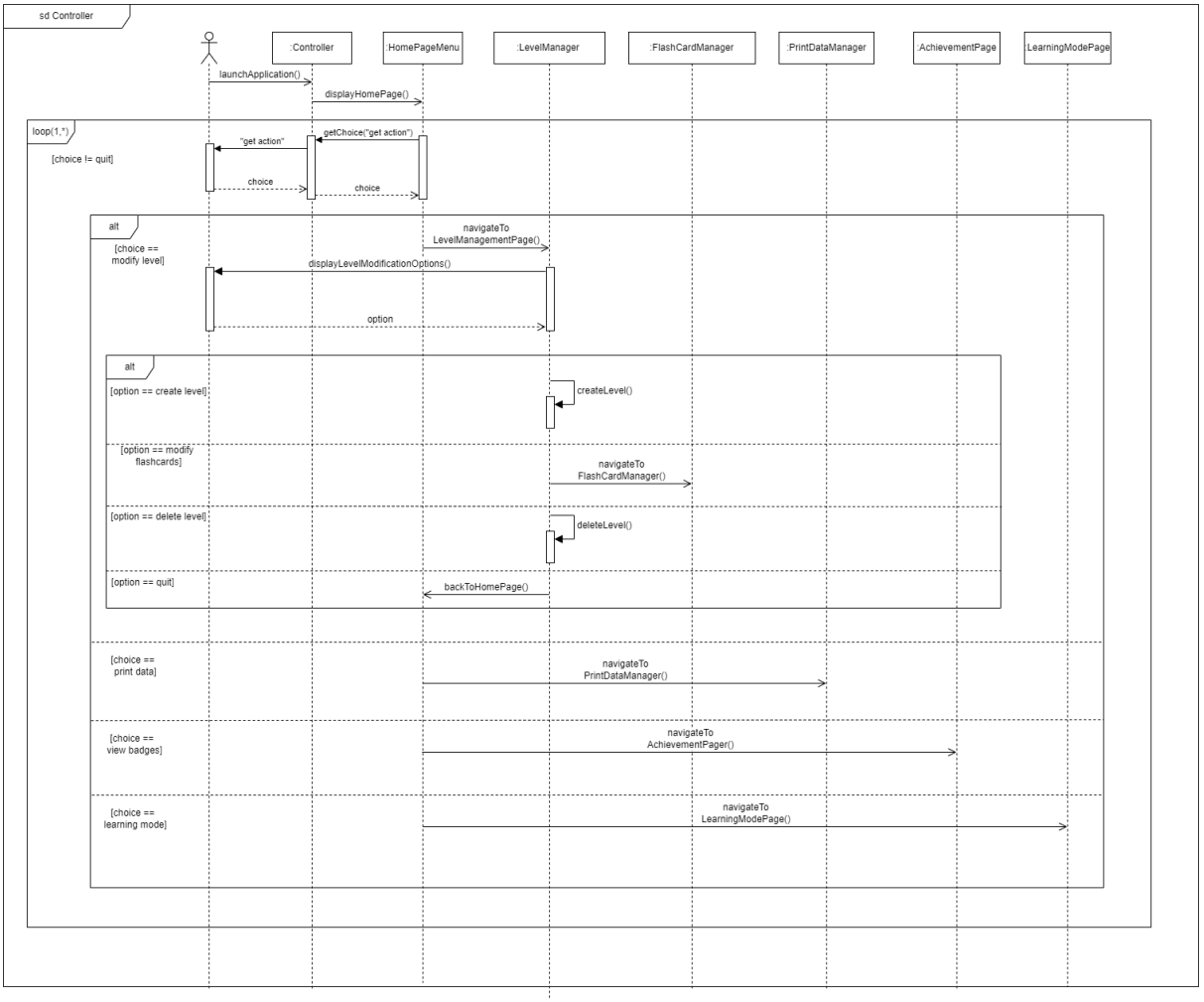
In the ***Mastered Flash Card*** state the system sets the card as mastered and also counts it for the progress and once the card is clicked as Mark as Mastered the word will skip and will not be shown in their learning sequence further.The inclusion of progress tracking, such as marking words as "mastered” and counting them for badges, adds a gamified element to the learning experience. This will enhance user motivation and engagement, turning the learning process into a rewarding journey.

The “LearningMode” state machine diagram ensures a structured and engaging learning experience, offering users the flexibility to review and mark words based on their understanding and preferences.

Revised sequence diagrams

Author(s): Jing Chen

Controller Options



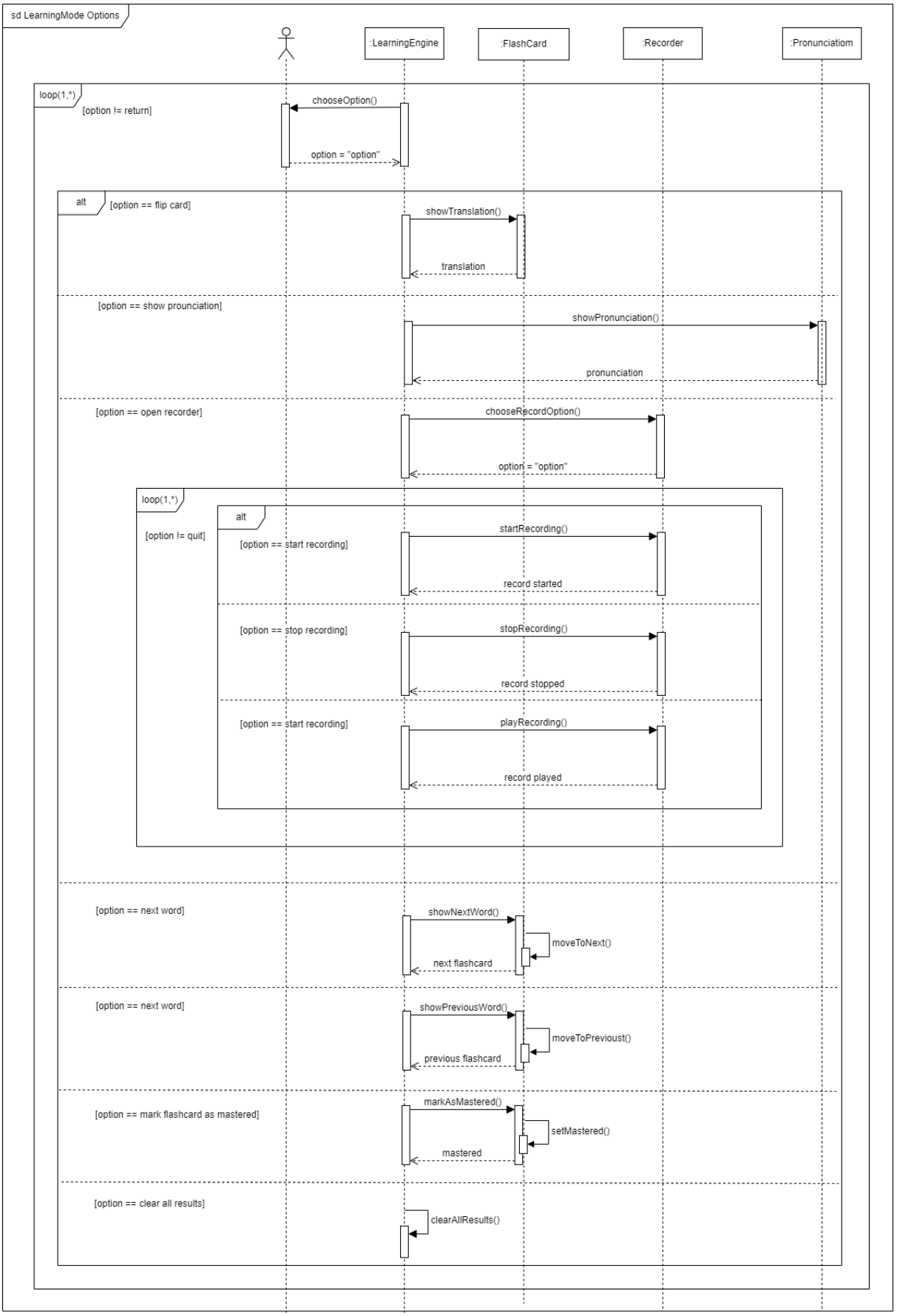
For this revised sequence diagram, we show the interaction that how users navigate between different pages from the home page, involving **User**, **Controller**, **HomePageMenu**, **LevelManager**, **FlashCardManager**, **LearningModePage**, **PrintDataManager**, and **AchievementPage**.

- **User**: The sequence diagram begins with the User launching the application, whereupon the Controller takes charge, directing the User to the HomePage.
- **Controller**: Manage the flow of interactions between the **User** and various pages. It receives input from the **User**, processes their requests, and directs them to the appropriate pages based on their selections.

- **HomePageMenu**: Acts as the starting point for the **User**'s journey within the application. It displays a menu of options available to the **User**, facilitating navigation to different functionalities.
- **LevelManager**: Provides tools for managing levels within the application. Users can create, edit, or delete levels as needed, contributing to the customization and organisation of their learning experience.
- **FlashCardManager**: **Users** can move to the FlashCardManager from the LevelManager. Here, they can perform actions such as adding, editing, or deleting flashcards.
- **LearningModePage**: Allows **Users** to engage in learning activities.
- **PrintDataManager**: Offers advanced features for importing external resources or exporting results.
- **AchievementPage**: Tracks **User** progress and achievements within the application. It showcases flashcard mastered and badges earned.

The revised sequence diagram introduces a more detailed perspective on how users interact with the application. The key modification is that we Integrated level management and flashcard management together. In the original diagram, users had the option to navigate separately to the **LevelManagementPage** and the **FlashcardManagementPage** directly from the **HomePage**. In contrast, the revised version simplifies this by only including a navigation option to the **LevelManager** within the **HomePageMenu**. Once users are in the **LevelManager**, they then have the option to proceed to the **FlashcardManager**. This modification allows for easy access and clear management of flashcards at any level, supporting personalised learning experiences.

LearningMode Options



The "LearningMode" sequence diagram below illustrates interactions involving **User**, **LearningEngine**, **Recorder**, **FlashCard**, and **Pronunciation** during the learning mode.

- **User**: Initiates the learning mode..
- **LearningEngine**: Facilitates the learning process, showing all the learning options to the users. Coordinates the flow, ensuring smooth transitions between different actions.
- **Recorder**: Provides audio support for recordings during the learning process.
- **FlashCard**: Contains the learning content, such as words and translations.
- **Pronunciation**: Provides audio support for pronunciations during the learning process.

The revised interaction begins with the **User** selecting a learning option to initiate the learning process. The **LearningEngine** then verifies the **User**'s chosen option. Within the primary loop of the sequence diagram, the **LearningEngine** awaits user input for various options. These options, including flipping the card, displaying pronunciation, opening the recorder, or marking flashcards as mastered, are presented using the "alt" operator. For instance, if the **User** selects "flip card," the **FlashCard** reveals the backside content through the showTranslation() method.

For this updated sequence diagram, we reconsidered the options available to users in the LearningMode. In the previous sequence diagram, users had the ability to flip flashcards, show pronunciation, record, review recordings, and mark flashcards as mastered. However, users can only navigate to the next or previous flashcard after flipping the flashcard. In the modified sequence diagram, we introduced two new options: show next word and show previous word. Consequently, users can navigate to the next or previous word even without flipping the flashcard. This adjustment aims to enhance users' learning experiences by providing additional flexibility in navigating through the learning content.

Another modification is the consolidation of the recording option and reviewing recording option. In the previous diagram, users could choose to record and review recordings separately within the LearningMode. However, in the revised version, we introduced the option of opening the recorder. Users now have the choice to both record their sound and review their recordings directly within the recorder. This modification streamlines the user experience by centralising recording-related actions into a single coherent option, simplifying the learning process and reducing potential confusion.

Implementation

Author(s): Yitong & Jing & Sree & Menghan

In this chapter, you will describe the following aspects of your project:

- the strategy that you followed when moving from the UML models to the implementation code;
- the key solutions that you applied when implementing your system, e.g., for especially challenging functionality or functionality where a lot of different options existed for the implementation;
- the location of the main Java class needed for executing your system in your source code;
- the location of the JAR file for directly executing your system;
- the link to the 30-seconds video showing the execution of your system (you are encouraged to put the video on YouTube or the video platform of your choice).

1. From UML to Code
   a. Main structure: Package diagram

      Before embarking on the implementation phase, we employed a package diagram to comprehensively visualise the project's architecture. This diagram served as a roadmap, providing a holistic view of the system's structure and aiding in the division of tasks among team members. By breaking down the project into distinct modules, including flashcard learning, audio recording and sound management, persistence handling, and level adjustment functionalities, we ensured a systematic approach to development.

      Moreover, the package diagram enabled us to identify potential dependencies and interactions between modules, guiding our design decisions and fostering modularity within the codebase. This approach aligns with software engineering best practices, where modular design facilitates code reuse, maintainability, and scalability.

   b. Detailed structure: Class diagram

      The class diagram serves as a pivotal tool in elucidating the architectural framework of a system, playing a significant role in its design. Firstly, leveraging the principle of Inversion of Control (IoC), our design adopts a clear separation of concerns: the FXML files define the UI layout, while the Controller classes handle user interactions and business logic. This approach enhances clarity in class categorization, facilitating a more systematic organization.

      Taking the Flashcard learning mode as an example, the design process commences with the formulation of the Flashcard class, encompassing all pertinent attributes and associated operations. Subsequently, within the Flashcard Study Mode class, operations pertaining to flashcard manipulation are encapsulated. These operations encompass functionalities such as loading and pars ing flashcard learning progress, initialising the flashcard study mode interface, and implementing fundamental learning operations, including flipping flashcards, marking flashcards as mastered, and navigating between flashcards. Furthermore, functionalities for clearing learning progress, employing the Persistence class for saving and loading learning progress, managing main page navigation, and handling exceptions are also incorporated.

The primary controller also includes the PrintController, which interfaces with fundamental classes like Persistence. Within the controller, key functionalities entail loading and displaying learning progress information from the "progress.json" file, reading information from the learning progress, printing learning data to JSON format files, importing new words from the "data/addData.json" file, merging them with existing data, and updating the learning progress file.

In the Modify Level controller, the architecture becomes relatively intricate, comprising two layers: adjusting levels and modifying flashcards within levels. In this segment, a novel approach is adopted, whereby each functionality is associated with a pop-up window, enhancing user operability from a user-centric perspective.

Efforts have been made towards enhancing the design of flashcard deletion and modification functionalities. Specifically, the incorporation of search functionality within deletion and modification operations is notable. Prior to deletion, the word's presence in the stored vocabulary is verified, ensuring deletion only occurs if the word exists in the vocabulary. In the modification mode, if the word is found in the vocabulary, its translation is automatically populated.

c. Overall interaction: Sequence diagram & state machine diagram

The controller sequence diagram provides a comprehensive overview of the main menu's interaction flow, illustrating how users navigate between different pages and return to the main menu. It showcases the sequence of events triggered by user actions, offering insights into the system's navigation logic.

The learning mode options illustrate the dynamic interaction between users and the learning environment. The sequence diagram reveals the intricate interactions among various classes or components involved in the learning process. It highlights how users interact with the system to check words, initiate sound mode, master words, and manage learning records.

Additionally, the state machine diagram complements the sequence diagram by depicting the states and transitions within the learning mode. It provides a visual representation of the user experience, outlining the possible states the system can be in and the events that trigger transitions between these states. This helps in understanding the flow of user interactions and system responses during the learning process.

2. Key solutions:

a. Object identity, equality, uniqueness：

At first, I didn't design the system with a unique identifier for each flashcard. Instead, I planned to use a combination of the level and ID as a unique identifier. For instance, both Level 1 and Level 2 could have an ID of 1, which meant that the ID was not unique on its own but was unique when combined with the level. However, I encountered two main issues with this approach:

1. It didn't meet my requirement for all words within a specific level to be arranged in numerical order. The deletion of specific flashcards could disrupt this order.
2. The algorithm became more complex, leading to confusion in assigning IDs.

Due to these challenges, I reverted to assigning a unique ID to each flashcard based on the order they were added to the database, disregarding the level. This approach ensures that each data entry has a distinct ID, eliminating concerns about duplicate IDs or confusion caused by one ID corresponding to different words.

Incorporating unique identifiers for objects, like flashcards, offers several benefits. It ensures clarity in identifying and distinguishing individual objects, facilitates accurate comparison and retrieval operations, maintains data integrity, and simplifies system logic and operations.

b. Inversion of Control: In JavaFX, the concept of Inversion of Control is realised through the use of FXML files and Controller classes. FXML files are responsible for defining the UI layout, while Controller classes handle user interaction and business logic.

   This approach greatly aids in modularizing and validating code to ensure it meets functional requirements. With FXML files defining the UI layout, I can verify the required and tested functionalities by running the code and observing the corresponding interface. For example, in my code scenario, we can illustrate this with the BadgeManagerUI class. In the BadgeManagerUI class, we define the UI layout and display logic. Simultaneously, the Controller class (i.e., the BadgeManagerUI class) is responsible for handling user interaction and business logic. The Controller can perform operations on the UI. This separation of UI layout and logic processing achieves the concept of Inversion of Control.

c. In our design process, we also applied concepts similar to the Observer pattern. For example, `progress.json` can be viewed as an observer reflecting the state changes of flashcards on `FlashcardStudyMode`. At the same time, the `PrintController` class observes the `progress.json` file and performs corresponding operations based on its data state. After `progress.json` changes, both `PrintController` and `BadgeManagerUI` can adjust accordingly.

d. In addition to utilising JavaFX, we employ the Gson library to facilitate the loading and saving of system states. Gson provides functions to interact with our JSON file. Upon reopening the application, the previous system state is restored by deserializing the data stored in the `data.json` file. When the user exits the application during its runtime, this file is updated to reflect the current system state by serialising the relevant information into JSON format.

e. UML has enhanced the clarity of our design. Initially, we did not create a state machine diagram for adjusting levels. Originally, we utilized several databases to store the flashcards we created or imported. However, this led to difficulties in synchronizing updates between databases. Additionally, due to imperfect ID assignment rules, we encountered situations where the same word had different IDs in different storage databases. To address this issue, our team revisited the use of a state machine diagram during discussions to streamline the process of modifying levels and flashcards, ultimately bringing clarity to the convoluted logic.

3. Notable Locations

Main Java Class	…\Software-Design-Group-112\src\main\java\nl\vu\cs\softwaredesign\Main.java
JAR File	…\Software-Design-Group-112\build\libs\software-design-vu-2024-1.0-SNAPSHOT.jar

4. The link to the demo of our application: https://www.youtube.com/watch?v=X9ZS2kQJiwY
(speed up version: https://www.youtube.com/watch?v=4S8Jntw-k0s)

Time logs

	A	B	C	D
1	**Member**	**Activity**	**Week number**	**Hours**
2	Jing Chen	Review feedback of assignment2	5	2
3	Yitong Tang	Review feedback of assignment2	5	2
4	Menghan Zhang	Review feedback of assignment2	5	2
5	Kota Sree Pragnya	Review feedback of assignment2	5	2
6	Jing Chen	Revise class diagram, object diagram, state machine diagram, package diagram,and sequence diagram	5	4
7	Yitong Tang	Revise class diagram, object diagram, state machine diagram, package diagram,and sequence diagram	5	4
8	Menghan Zhang	Revise class diagram, object diagram, state machine diagram, package diagram,and sequence diagram	5	4
9	Kota Sree Pragnya	Revise class diagram, object diagram, state machine diagram, package diagram,and sequence diagram	5	4
10	Jing Chen	Implement and test the system, make UI design	6	12
11	Yitong Tang	Implement and test the system, make UI design	6	12
12	Kota Sree Pragnya	Implement and test the system, make UI design	6	12
13	Jing Chen	Write up feedback points, make the final report	7	10
14	Yitong Tang	Write up feedback points, make the final report	7	10
15	Menghan Zhang	Write up feedback points, make the final report	7	10
16	Kota Sree Pragnya	Write up feedback points, make the final report	7	10
17				
18			**TOTAL**	100