Assignment 2

Team number: 112
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 1

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

Based on the feedback from last time and after reviewing peers' work, we have identified two areas for improvement: 1) provide more detailed function design, 2) apply better methodology to articulate why we design the system in a certain way. To enhance the efficiency of the assignment, we will employ methods such as user stories and protocols to develop a more specific design and provide additional details to the system.

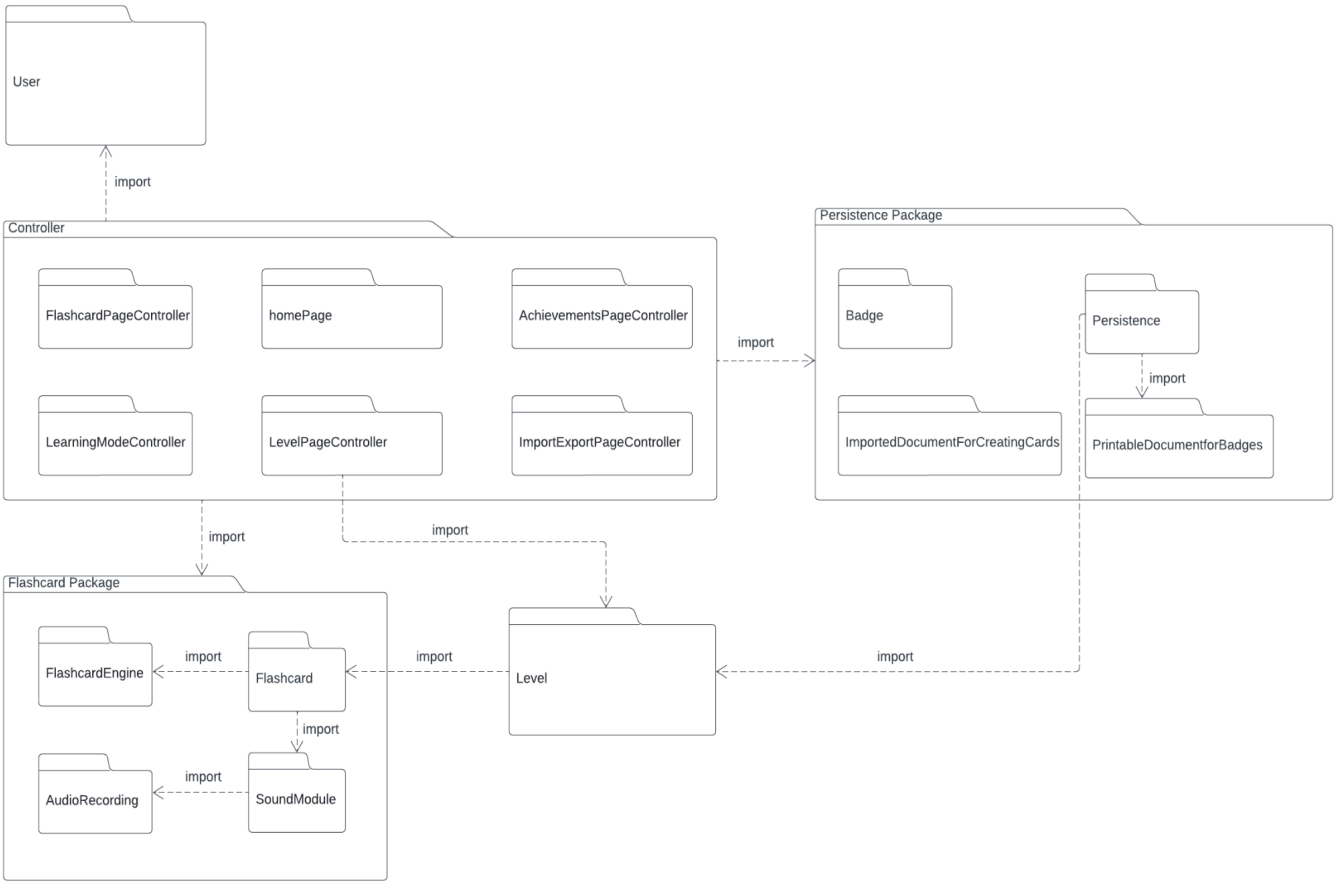
- Feature 3: the Persistence component has been upgraded to provide more detailed storage and management for levels and flashcards. Each JSON file now includes comprehensive information about levels and flashcards, such as the level's name, difficulties, description, and flashcard details. These enhancements ensure effective capture and management of learning content for the future developing work.
- Feature 4: this time we include word display in target language, mastery marking, and pronunciation features. We try to use detailed design to facilitate efficient language acquisition and personalised learning paths. In setting the level boundaries, such as the 100% threshold, and refining the functionality of the flashcards, we have enhanced the system for subsequent implementation.
- Feature 5: we have refined the design details of badges, incorporating both custom and automated designs. This modification facilitates greater user engagement.
- Feature 6: we've designed a semi-automated word generation feature. A hidden language list will be stored in the database alongside pre-designed levels. Users can access the word list corresponding to their level and select words to add to their learning level. This approach efficiently extracts relevant words from the database.

- Feature 7: we've clarified our approach to handle audio-related tasks through APIs and have prepared the necessary database and technical documentation for reference.
- Feature 8: we've specified the data printing format as JSON, which allows for easier development and execution by utilizing external frameworks to convert it into strings.
- Feature 9: we've established the following rule: Before importing data, the system will verify whether the input data meets JSON format requirements and includes the word, translation, and level number, facilitating better data validation.

After all these revisions, we'll utilise the MoSCoW method to prioritise features and allocate tasks for software implementation.

Package diagram

Author(s): Menghan Zhang



According to the class diagram, the package diagram is divided into five main sections: “User”, “Controller”, “Flashcard Package”, “Persistence Package” and “Level”. The package diagram is package by feature.

Controller package contains five sub controller packages and homepage. **Controller package** imports **User package**, as Controller class invokes User class in the class diagram.

Flashcard package contains four packages about flashcards. **Flashcard** imports **FlashcardEngine** and **SoundModule**, as Flashcards use other two modules. **SoundModule** imports **AudioRecording** in the same way.

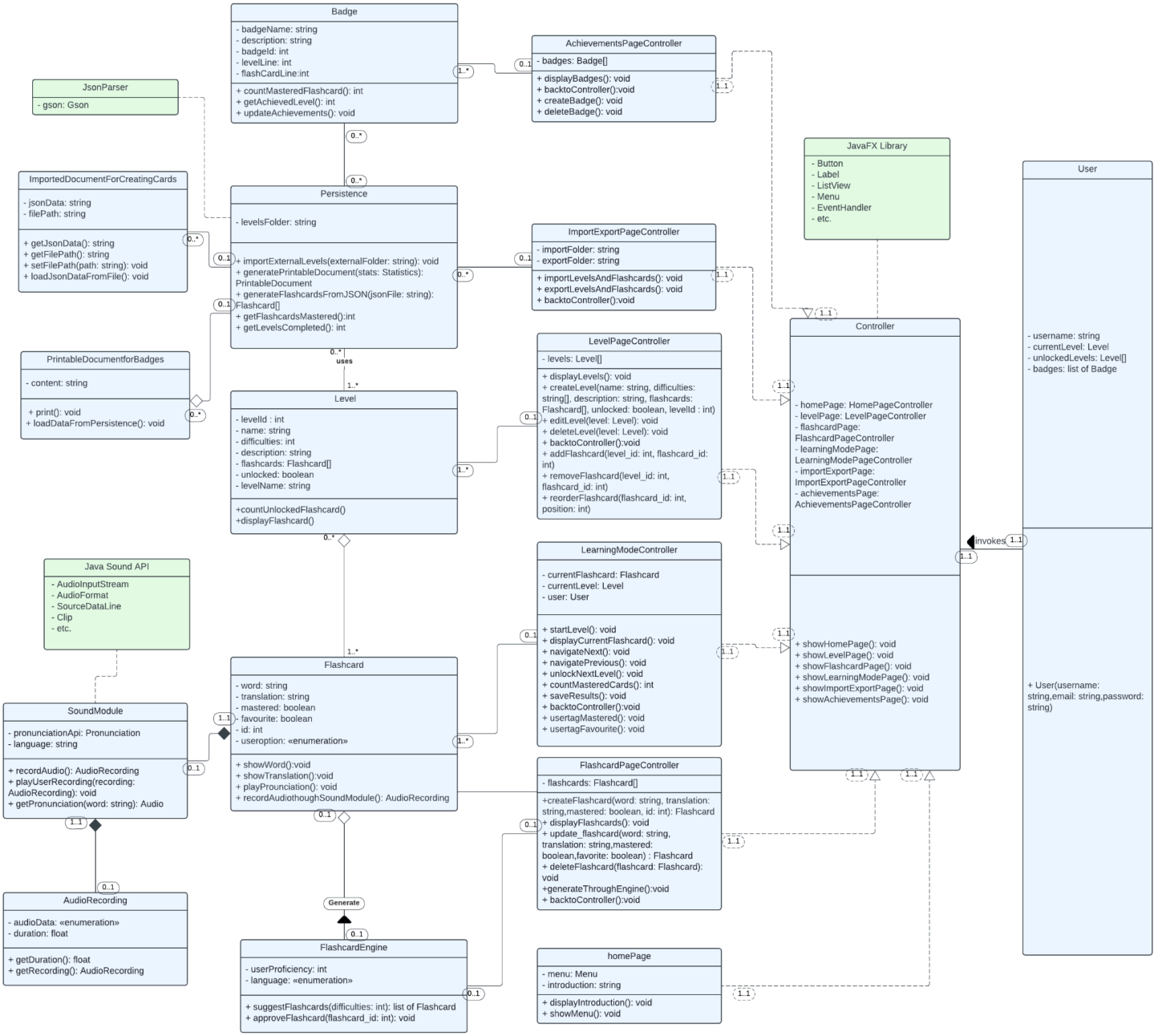
Level package imports **flashcard package**, as Level package is combined with Flashcard package. **LevelPageController package** imports **Level package**, as it is the controller of level.

Persistence package contains four packages about persistence. Persistence class uses Level class in class diagram, so **Persistence package** imports **Level package**. Persistence also imports **PrintableDocumentforBadges**, as Persistence class and PrintableDocumentforBadges class are aggregation in class diagram.

As Flashcard package and Persistence package are used in Controller package, **Controller package** imports **Flashcard package** and **Persistence package**.

Class diagram

Author(s): Yitong Tang



Design Method:

We aimed to 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 four parts, presented from right to left. Although layout consideration isn't mandatory in the class diagram, we find it aids in understanding the structure comprehensively.

1. Detailed Functions Linked with Main Class:

These classes, such as SoundModule, and PrintableDocumentforBadges, 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.

2. 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().

3. 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().

4. User and Controller:

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

Here is the further explanation of the classes

User: Represents a user of the system.

Attributes:basic information and the levels the user is currently studying(currentlevel), unlockedLevels stores a list of levels the user has unlocked, and list of badge they have achieved

Operations: creating a new user

Associations: associates with controllers which act as the main intermediaries with all the function mode.

Controller: Manages the different pages of the GUI and their controllers, As an intermediary for transitioning between different controllers, it is also an Interaction Interface. I had considered using the homepage as an intermediary, but in order to separate the functionalities of each model, I decided to establish a controller to manage the overall process.

HomePageController: Controls the Home Page, including introduction and menu.

Operations: from the home page, users can choose to enter in the function module that they are interested in, the displayIntroduction() is used to show the system introduction.

Associations: Composition relationship with Controller, the controller will act as the intermediaries.

FlashcardPageController: Controls the Flashcard Management Page, including displaying flashcards and providing options to add, edit, and delete flashcards.

Attributes: an array of flashcard that the system have in the database

Operations: functions to create, update, delete flashcards manually, such as +createFlashcard()

+ displayFlashcards() is used for the user to check the flashcards they already have.

+generateThroughEngine() is used for generating flashcards half automatically though the database we already have.

+ backtoController():back to the home page to choose another mode.

Associations:

- Association with Flashcard class for flashcard editing. - Composition relationship with Controller

LearningMode: Represents the learning mode functionality where the user can study flashcards.

Attributes:- currentFlashcard: Represents the flashcard currently being studied.

- user: Represents the user who is learning the flashcards.
Operations: stands for the functions that the user can take when they start the learning mode:
+ startLevel(): Start from the level that is already saved.
+ displayCurrentFlashcard():Display the current flashcard being studied.
+ navigateNext(): void: Navigate to the next flashcard.
+ navigatePrevious(): void: Navigate to the previous flashcard.
+ countMasteredCards(): count the mastered cards of the user when leaving the game, this will contribute to the achievement part.
+ saveResults(): save the result of the game.
+ wordMastered(): Calculates the number of mastered words.
Associations:
- Composition relationship with Controller; - Association with Flashcard class for learning content.
LevelPageController: Controls the Level Management Page, including displaying levels and providing options to add, edit, and delete levels. For this part, the main function and the structure is quite close to the class of **FlashcardPageController.** The different function it has is it can not only control level, but also control the flashcard from a higher level, removeFlashcard():removes flashcards from the level and reorderFlashcard is for arranging the order.
The following two classes are straightforward and their relationships are similar to other controllers mentioned above, with functionalities directly pointing to relevant functional classes: **ImportExportPageController:** Controls the Import/Export Page, including specifying import and export folders and handling the import/export operations. **AchievementsPageController:** Controls the Achievements Page, including displaying earned badges.

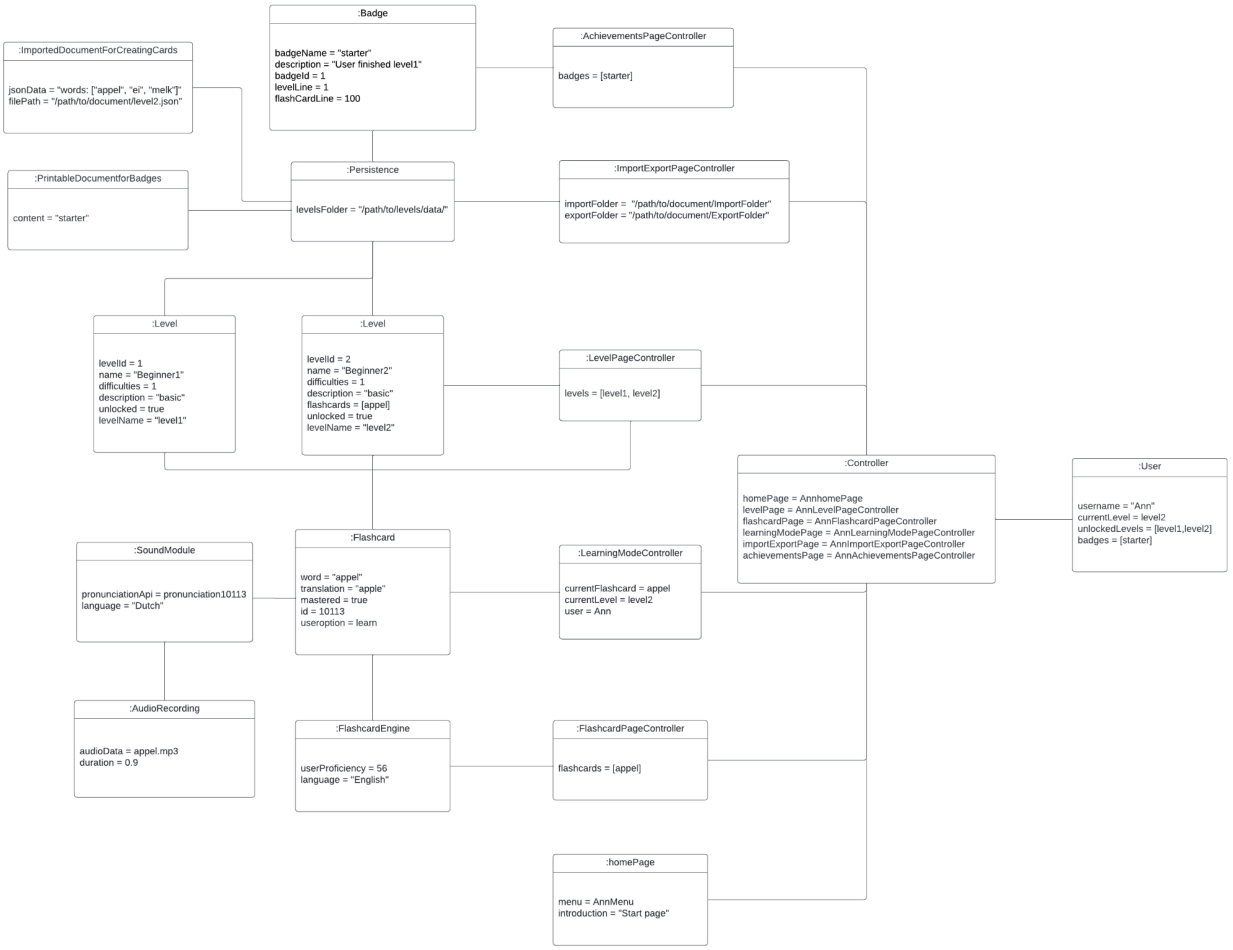
The following are important classes and their detailed functions:

Flashcard: Represents a flashcard containing word, translation, ID, and tag information. Attributes contain the basic information of a word, and also include the tag given from the user.
Operations: Basic functions of a flashcard ,such as showing two sides: showWord() and showTranslation(), also includes sound modes such as playPronunciation(): Plays the pronunciation audio, and recordAudioThroughSoundModule(): Records audio.
Associations: - Composition relationship with Pronunciation for pronunciation functionality. - Composition relationship with AudioRecording for recording audio.
Level: Represents a learning level within the system.
Attributes: level's name, difficulty level, description, whether it is unlocked and an array of flashcards(an array of Flashcard objects representing the flashcards associated with the level)
Operations: +displayFlashcard()is used to show all the flashcards in the function.
+ countUnlockedFlashcard(): Counts the number of unlocked flashcards.
Associations:
- Aggregation relationship with Flashcard class representing associated flashcards.
Persistence: The component responsible for persisting levels and flashcards as JSON files
Attributes: levelsFolder: Stores the path to the folder where level JSON files are stored.
Operations: + importExternalLevels(): Allows users to import external JSON files containing levels into the system.
+ exportLevels(): Enables exporting existing levels to a specified folder.
+ generateFlashcardsFromJSON(): Generates flashcards from a JSON file.
+ getFlashcardsMastered(): Returns the number of mastered flashcards.

+ getLevelsCompleted(): Returns the number of completed levels.
Associations: - Associated with ImportedDocumentForCreatingCards class for importing/exporting operations;- Associated with PrintableDocumentforBadges for printing documents in JSON format.
Badge: Represents an achievement badge earned by the user.
Operations: includes creating customised badges such as createBadge() deleteBadge() and also generate automatically based on the master flashcards, such as countMasteredFlashcard(): Calculates the number of mastered flashcards associated with the badge.
Associations:
- Associated with Persistence for badge creation; - Associated with Flashcard class as badges are developed from mastered cards.
SoundsModule: Responsible for integrating audio functionalities into the system.
Attributes: - pronunciationApi: Instance of a Pronunciation class responsible for fetching correct pronunciation audio. - language: language for which pronunciation assistance is provided.
Operations: +playAudio(): Play audio from a given URL; recordAudio(): Record user's voice; playUserRecording(): Play back user's recorded voice; getPronunciation(word: string): Fetches correct pronunciation audio for a given word.
AudioRecording: Represents a recorded audio clip.
Attributes: -audioData: Data representing the recorded audio; -duration: Duration of the recorded audio clip in seconds.
Operations: + getDuration(): Retrieves the duration of the audio clip.
+ getRecording(): Records audio from the user.
FlashcardEngine: Generates flashcards based on the user's proficiency level and learning goals.
Attributes: -Language: Specifies the language for flashcards generation; -User proficiency: Stores the user's proficiency level.
Operations: +suggestFlashcards(): Suggests new flashcards based on user's proficiency level; +approveFlashcard(): Approves a suggested flashcard, adding it to the user's learning trajectory.
Associations: Aggregation relationship with Flashcard class representing flashcards.
PrintableDocument:A printable document containing learning results, mainly the badge.
Attributes: Content: Content of the document.
Operations: + loadDataFromPersistence(): Load the results of the mastered word, badges in json for printing. + print(): Print the document.
ImportedDocumentForCreatingCards: Represents the document imported from the JSON file.
Attributes: - filePath: Represents the path where the JSON file is stored.
- jsonData: Stores the data imported from the JSON file.
Operations: +getJsonData(): Retrieves the JSON data; setFilePath(): Sets the file path. +getFilePath(): Retrieves the file path; +loadJsonDataFromFile(): Loads the JSON data from the file specified by filePath.

[optional] Object diagram

Author(s): Menghan Zhang



According to the class diagram, we draw an object diagram that shows the user learning a new flashcard.

User object with name = “Ann " is the current user. She is learning a word from level2 right now. User’s unlockedLevels = [level1, level2], and User’s currentLevel = “level2”. User’s badge = “Starter”, which is obtained from **AchievementsPageController object**.

Controller object links to User object and other six sub controller objects, they are homePage, FlashcardPageController, LearningModeController, LevelPageController, ImportExportPageController and AchievementsPageController. Controller object summarises the functions of other sub controllers.
HomePage object shows the start page.

Ann is now learning “appel”, so the **Flashcard object** has Word = “appel " , translation = “apple”, id = 100113. Useroption = learn means Ann chooses to learn the word rather than skip the word. She

learned the new word, thus masters = true. **LearningModeController object** links to Flashcard object. It shows currentFlashcard = “appel”. This word belongs to level2, so currentLevel = level2.

SoundModule object links to Flashcard object. PronunciationApi = pronunciation10113, SoundModule object’s language is the source language, so language= “Dutch”. The **AudioRecording object** is related to the SoundModule object. audioData = appel.mp3, the duration = 0.9.

FlashcardEngine object links to Flashcard object. UserProficiency = 56, FlashcardEngine object’s language is the target language, thus language= “English”.
FlashcardPageController object controls FlashcardEngine object, flashcards = “appel”.

Level object manages Flashcard object. There are two level objects. The first Level object is level1, levelId = 1, name = “Beginner1”, difficulties =”1”, description = “basic”, unlocked = “true”, levelName = “level1”, while the second Level object is similar to the first one, except its levelId = 2, name = “Beginner2”, levelName = “level2”. As the user is learning in level2, this Level object shows the flashcards = “appel”.
Level objects link to **LevelPageController**, levels = [level1, level2].

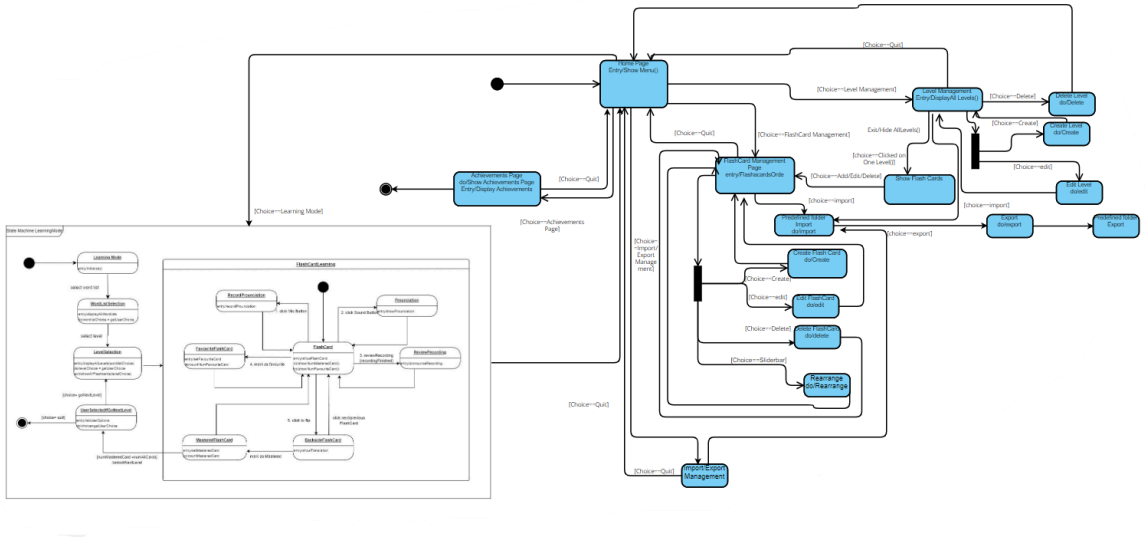
Persistence object masters Level objects, levelsFolder = "/path/to/levels/data/".
Badge object links to Persistence object, badgeName = "starter", description = "User finished level1",badgeId = 1, User finished level1, so levelLine = 1. One level contains 100 flashcards, flashCardLine = 100. **AchievementsPageController** shows the badge users have earned. In this case, Badges = starter.

ImportedDocumentForCreatingCards object links to Persistence object, jsonData = "words: ["appel", "ei", "melk"]", filePath = "/path/to/document/level2.json". **PrintableDocumentforBadges object** also links to Persistence object, content = “starter”.
ImportExportPageController links to Persistence object, importFolder = "/path/to/document/ImportFolder" and exportFolder = "/path/to/document/ExportFolder".

State machine diagrams

Author(s): 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 go back to the home page. 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 Level Management Page, when users enter they will encounter several states. In the ***Levels Displayed*** state, users can view all available levels. They can then seamlessly transition to the Adding Level, Editing Level, and Deleting Level states, depending on their actions to manage levels. Clicking on a specific level takes the user to the ***Flashcards Displayed*** state, where associated flashcards are visible and 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 Flashcard Management Page introduces users to the ***Flashcards Displayed*** state, showcasing all flashcards. Users can then transition to the Adding Flashcard, Editing Flashcard, and Deleting Flashcard states based on their interactions to manage flashcards and also rearrange the cards. Showing all flashcards on the Flashcard Management Page when a level is selected ensures easy

access and clear management. Allowing users to rearrange flashcards supports personalised learning experiences, accommodating diverse study preferences and improving overall usability which is what users expect in the learning mode.

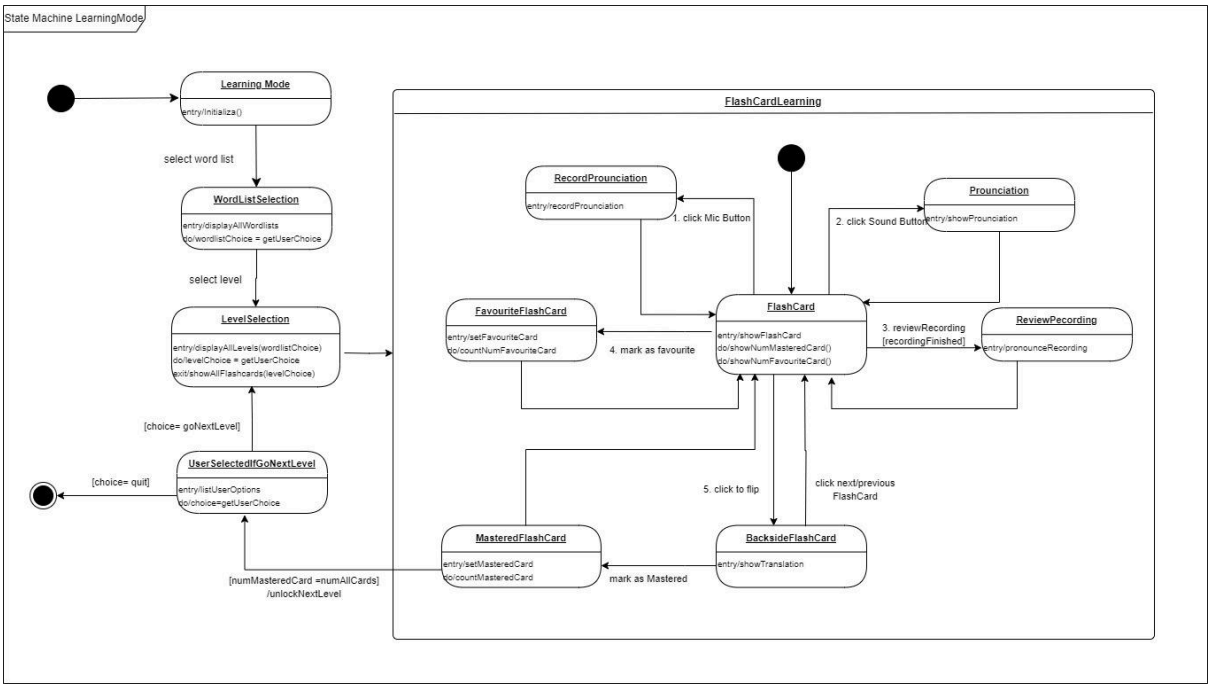
The Learning Mode Page presents the ***Flashcard Displayed*** state, where users engage with the current flashcard. We will give a detailed introduction of this part in our next state machine diagram, named “LearningMode”.

The Import/Export Page starts when the users make a choice when they want to import/ export .When users initiate import/export operations, they transition to the ***Importing/Exporting*** state where they import first and all the information goes to a predefined folder and then continue exporting.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 Achievements Page offers the ***Achievements Displayed*** 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 gets the user's choice. After the user choice is given the system exits from this state and then goes to the next state. 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 ***Level Selection*** state it displays all the levels to the user and allows the user to select the level.Initially when 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.Once the user gives the choice the flashcards of that level are shown and then the user can choose one flash card.We have designed this way as to allow users to select levels dynamically because it supports personalised learning experiences. This design decision also recognizes that users may have varying proficiency levels and learning goals so it will cater the needs of both beginners and advanced learners, fostering inclusivity.

When One flash card is selected it enters Flashcard state where there are several transitions. In this state it displays the flash card front where by clicking the sound button it enters to the ***pronunciation*** state in which the user can listen to the pronunciation of the word and by clicking the mic button the system will enter to the ***record pronunciation*** state where the user can record their

pronunciation and by clicking on the review recording the system will enter into the ***Review Recording*** state where the user can 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 exit the show card front() and click on click to flip the flashcard the system will enter into the ***BackSideFlashCard*** state where the backside of the flashcard is displayed showing the translation and meaning of the word .In this state it self after viewing the backside of the flash card the user has an option to go to the next card if the choice is next and if the choice is previous 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 and meaning of the word and feels that they are confident with the meaning and 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 self-assess their confidence themselves 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.

If the user has marked any of the words as "Favourite” the system enters into ***Favourite FlashCard*** state.Upon entry in this state the words are set as favourite and are counted for badges in later stages. The inclusion of progress tracking, such as marking words as "Favourite" 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.

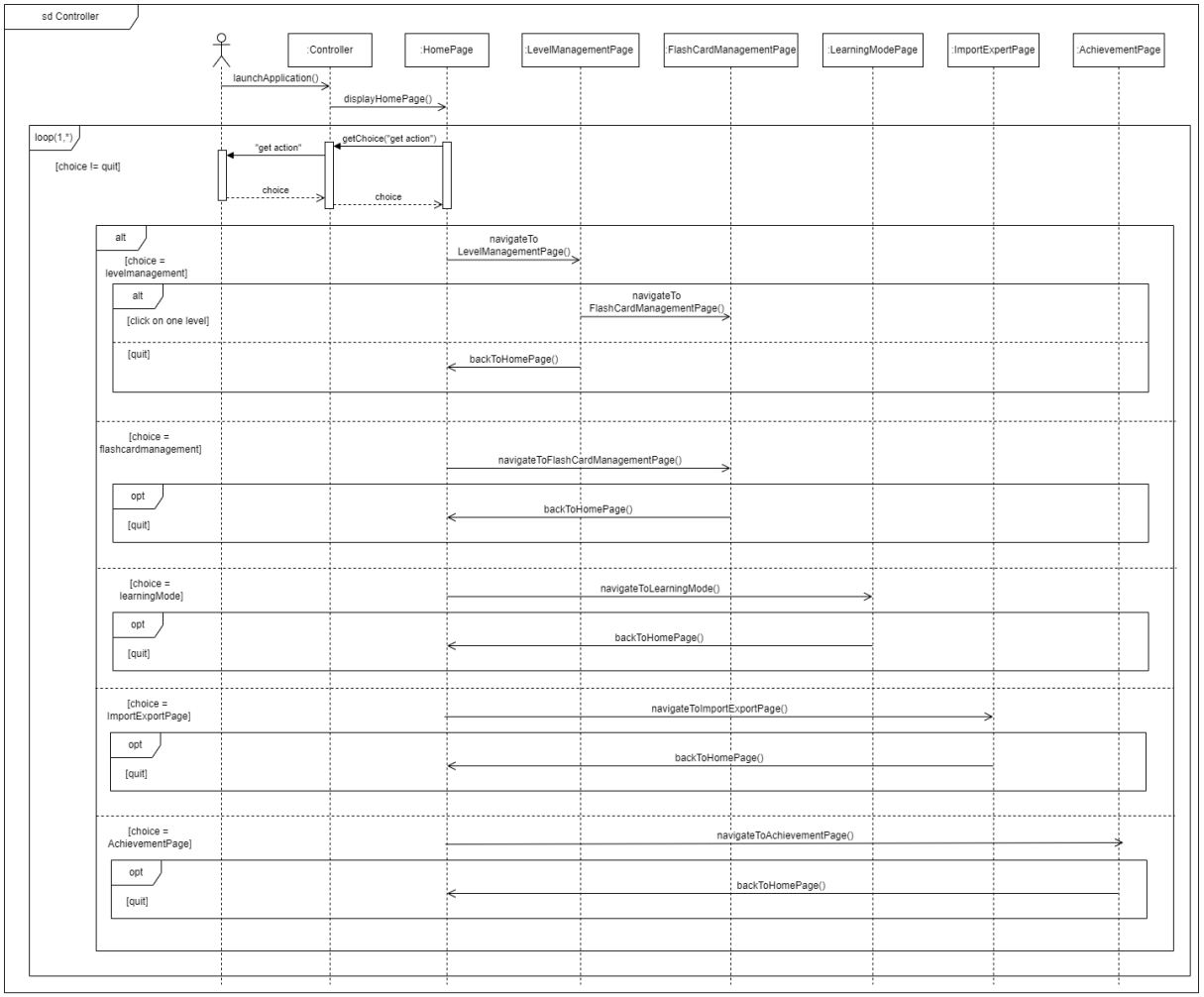
When the user completes all the flashcards as “mastered” the system will enter into ***User Selected*** to Go to the next level. In this state upon entry the user will be shown the list of options whether they want to go to the next level or quit and come back later as per their choice.This way we will acknowledges individual learning paces and preferences and also will allow users to challenge themselves by progressing or take breaks, fostering a learner-centric approach.

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.

Sequence diagrams

Author(s): Jing Chen

Controller Options



The "Controller" sequence diagram illustrates how users navigate between different pages in our application, involving **User**, **Controller**, **HomePage**, **LevelManagementPage**, **FlashCardManagementPage**, **LearningModePage**, **ImportExpertPage**, and **AchievementPage**.

- **User**: Initiates the interaction by launching the application. Throughout the sequence, the User makes selections from menus presented by the **Controller** to navigate between different pages.
- **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.

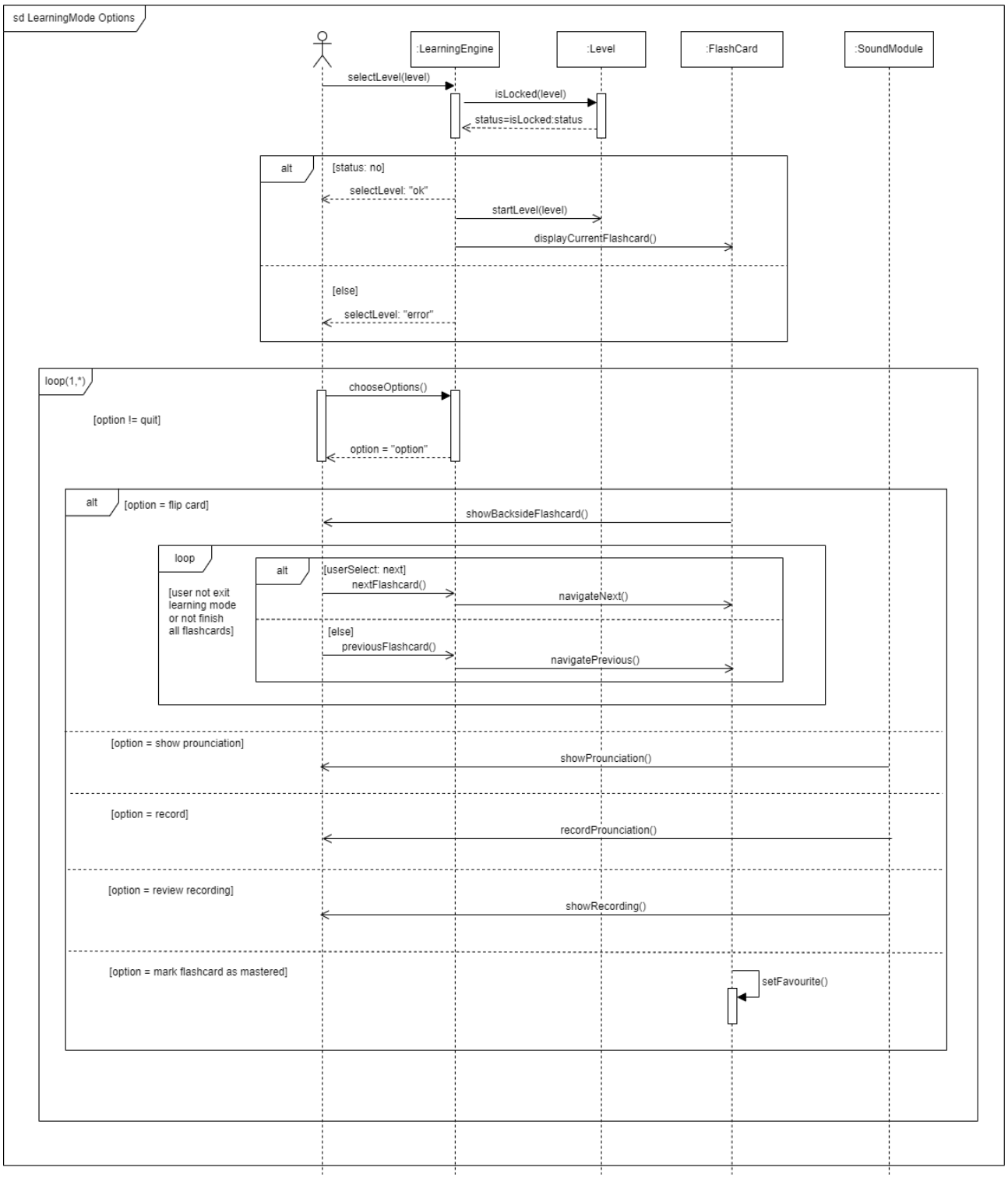
- **HomePage**: Acts as the starting point for the **User**'s journey within the application. It displays a menu of options available to the **User**, facilitating navigation to different functionalities.
- **LevelManagementPage**: Provides tools for managing levels within the application. Users can create, edit, or delete levels as needed, contributing to the customization and organisation of their learning experience.
- **FlashCardManagementPage**: Offers functionalities for managing flashcards. Users can add new flashcards, review existing ones, or organise them into categories, enhancing their learning process.
- **LearningModePage**: Allows Users to engage in learning activities.
- **ImportExportPage**: 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 interaction begins with the **User** launching the application. The **Controller** then displays the **HomePage**, presenting a Menu for accessing other pages. This initiates the main loop of the “Controller” diagram. Within this loop, the **User** is prompted to make a choice through returning message “choice”, such as navigating to the **LevelManagementPage**, **FlashCardManagementPage**, and so forth. The loop continues until the User chooses the "quit" option.

If the **User** opts to navigate to the **LevelManagementPage**, for example, this page is displayed. From there, the **User** can further navigate to the **FlashCardManagementPage** by selecting a level or return to the **HomePage** by choosing the "quit" option. To depict these branching possibilities, the "alt" operator is employed, showcasing the different paths users can take.

Similarly, for other choices like navigating to the **FlashCardManagementPage**, the "opt" operator is used, as it presents only one option (returning to the **HomePage**). This ensures clarity in representing the flow of interactions between the **User** and various pages controlled by the **Controller**.

LearningMode Options



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

- **User**: Initiates the learning mode by selecting a level to start.

- **LearningEngine**: Facilitates the learning process, checking if the selected level is locked. It returns "error" if locked, otherwise, returns "ok" and initiates learning, displaying the current flashcard.
- **Level**: Represents the selected level for learning.
- **FlashCard**: Contains the learning content, such as questions and answers.
- **SoundModule**: Provides audio support for pronunciations or recordings during the learning process.

The interaction begins with th **User** selecting a level to start learning. The **LearningEngine** verifies if the chosen level is accessible. If locked, it sends an "error" message; otherwise, it sends "ok" and proceeds with learning, displaying the first flashcard.

Within the main loop of the sequence diagram, the **LearningEngine** awaits user input for various options. These options, including flipping the card, displaying pronunciation, recording, reviewing recordings, 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 showBacksideFlashCard() method. This action triggers another loop, which continues until the **User** chooses to exit or completes all flashcards in the level.

Within this loop, an alternative interaction is provided using the "opt" operator, allowing Users to choose between viewing the next or previous flashcard. This flexibility enhances the learning experience by accommodating User preferences and pacing.

Throughout the interactions, the **LearningEngine** coordinates the flow, ensuring smooth transitions between different actions and providing a seamless learning experience for the **User**.

Time logs

<Copy-paste here a screenshot of your time logs - a template for the table is available on Canvas>

Team number	112		
Member	**Activity**	**Week number**	**Hours**
Jing Chen	Test UML environment, draft classes for class diagram, review feedback	3	4
Yitong Tang	Test UML environment, draft classes for class diagram, review feedback	3	4
Menghan Zhang	Test UML environment, draft classes for class diagram, review feedback	3	4
Kota Sree Pragnya	Test UML environment, draft classes for class diagram, review feedback	3	4
Jing Chen	Create drafts of class diagram, object diagram, state machine diagram, package diagram,and sequence diagram for review by mentor	4	6
Yitong Tang	Create drafts of class diagram, object diagram, state machine diagram, package diagram,and sequence diagram for review by mentor	4	6
Menghan Zhang	Create drafts of class diagram, object diagram, state machine diagram, package diagram,and sequence diagram for review by mentor	4	6
Kota Sree Pragnya	Create drafts of class diagram, object diagram, state machine diagram, package diagram,and sequence diagram for review by mentor	4	6
Jing Chen	Work on class, object, package, sequence, and state machine diagrams, discuss together	5	10
Yitong Tang	Work on class, object, package, sequence, and state machine diagrams, discuss together	5	10
Menghan Zhang	Work on class, object, package, sequence, and state machine diagrams, discuss together	5	8
Kota Sree Pragnya	Work on class, object, package, sequence, and state machine diagrams, discuss together	5	8
Jing Chen	Write up feedback points, review all diagrams and write report	5	6
Yitong Tang	Write up feedback points, review all diagrams and write report	5	8
Menghan Zhang	Write up feedback points, review all diagrams and write report	5	6
Kota Sree Pragnya	Write up feedback points, review all diagrams and write report	5	6
		TOTAL	102