
Eötvös Loránd University

Faculty of Informatics

Department of Computer Algebra

# Melodious: Web application for learning/teaching musical instruments

*Supervisor:*
Dr Harrison-Juhász Zsófia
Lecturer

*Author:*
Akimzhanov Zheenbek
Computer Science BSc

*Budapest, 2023*

## Original Thesis Declaration

The recent pandemic resulted in education/work transitioning into online format through special web pages or applications which brought a realisation to many people that knowledge and experience can be gained not only by actually attending in person. All the online platforms and services finally got acknowledged by a bigger number of people. Personally, I started learning guitar by watching YouTube tutorials and by doing some readings in different pages. I noticed that most of the musical instrument tutors offer their online courses through their own external web pages. So the problem is when it comes to learning musical instruments remotely, there are not great varieties of courses bundled together in one place. And I thought: "Why is there not any platform specifically dedicated for different music tutors to upload courses for people to buy if interested?". So I decided just to do that.

The main goal of the project is to provide a public music learning/teaching platform with its own facilities and features which make it easier for the learners to find the needed course and learn conveniently, and for the music tutors to be able to publish their own courses and manage their accounts. The platform should provide a way to maintain a communication between tutors and learners and be able to provide ways of interactive teaching/learning.

The basic features of the web application are going to be:

- Being able to register the user
- Being able to both publish and view others' content
- The format of the lessons may include video, text, note, sheets/tabs, etc.
- The musical instruments are categorised into different sections and can be easily navigated through
- Tutors and learners can communicate
- The progress is saved throughout the course

The application can be extended way further with the features uniquely related to instruments. For the implementation ReactJS, JavaScript and probably Java will be used. Some Python libraries might also be needed.

# Contents

# Chapter 1

# Introduction

## 1.1 The idea and the motivation behind the project

The idea of a unified platform for the music teachers'/learners' community was sparked by a realization during the worldwide transition to online education: a digital platform could be a thriving hubs for knowledge, especially in the realm of music education. However, the online landscape for learning musical instruments are separated across different platforms and personal pages, resulting in scattered resources. So the idea is to bridge this gap by providing a centralized, community-driven platform for music lovers.

It's where tutors and learners can fluidly switch roles and cultivate a culture of shared knowledge and experience. Every user can share a course or take a course. The platform stands out for its user-friendly interface and unparalleled effectiveness, mirroring the dynamics of social media allowing users to connect with each other and chat but with a focus on music themes.

It goes beyond simply consuming content by encouraging active contributions from its users, elevating the overall learning experience. Communication and social collaboration are at the core of this platform, fostering a vibrant ecosystem for music learners to thrive in.

Melodious was intentionally developed to accommodate future growth and further enhancement. Its innovative software design is able to anticipate and cater to the changing demands of the music education community, making it a potential central hub for enthusiasts and educators worldwide. Beyond being a mere educa-

tional tool, Melodious fosters a dynamic and constantly evolving environment for the global music community to come together, learn, and collectively develop.

## 1.2 Project Overview

Melodious, an innovative online platform, blends teaching and learning of musical instruments in a community-driven environment. It offers a flexible system for creating and categorizing courses with diverse, customizable content types like text, videos, images, and document files. This adaptability enriches the learning experience, catering to varied styles and preferences. Highlighting core functionalities such as course creation, communication, and community engagement, Melodious ensures a comprehensive and interactive musical education journey.

**Course Creation and Management**: Users can create, edit, and categorize courses which can be free or paid. Courses can include various content types such as text, images, videos, and quizzes. Courses can be in a draft state or published.

**Communication Features**: The platform supports user-to-user messaging with the ability to send attachments, fostering community engagement and collaboration.

**Learning and Exploration**: Users can enroll in courses, save them for later, and search for courses and users. The platform also recommends courses based on user activity.

**User Interactions and Reviews**: Users can leave reviews and ratings for courses and other users, enhancing the community aspect and providing feedback.

**Community Engagement**: Users can follow each other, creating a community atmosphere where they can keep track of their favorite educators or fellow learners.

# Chapter 2

# User Documentation

Welcome to the User Documentation of Melodious, an engaging and innovative platform for music education developed within the frames of this thesis. The web site is available at https://melodiousacademy.com. This chapter provides a comprehensive guide to navigating and utilizing the various features of Melodious (here: 3.1). Should the website be unavailable, please refer to the Installation Guide in the Developer's Documentation for local setup instructions. Here, you will find detailed descriptions of the homepage layout, user authentication process, course exploration and management, transaction handling, and other key functionalities that enhance the user experience on Melodious.

## 2.1 Homepage Overview

The Melodious homepage (https://melodiousacademy.com) offers a welcoming overview, accessible without authentication. It features:

- **Navigation Bar:** Includes Features, Courses, Feedback, and Dashboard options.
- **Introduction Section:** Highlights the platform's core values and a 'Get Started' button, which leads unregistered users to the login/register page.
- **Main Features Section:** Major features are listed here along with tile of pictures for visualising the mentioned feature. Currently it includes the fol-

lowing feature titles: "Create Your Music Course", "Explore And Learn" and "Connect & Collaborate".

- **Course Browsing Section:** (refer to Figure 2.1 below) Users can explore courses by categories or through direct search. Clicking on a category icon shows all courses within that category. This section also provides some of the most enrolled courses in the platform.
  - All courses from this section, whether sourced from search, category selection, or most enrolled courses, are viewable by all users, including unauthorized ones.
  - Course descriptions are accessible to unauthorized users, but enrolling in a course redirects them to the login/register page.
- **User Testimonials and FAQs:** Provide insights into user experiences and common inquiries.
- **Enrollment Process:** Viewing courses is unrestricted, but enrolling requires redirection to the login/register page for authentication.

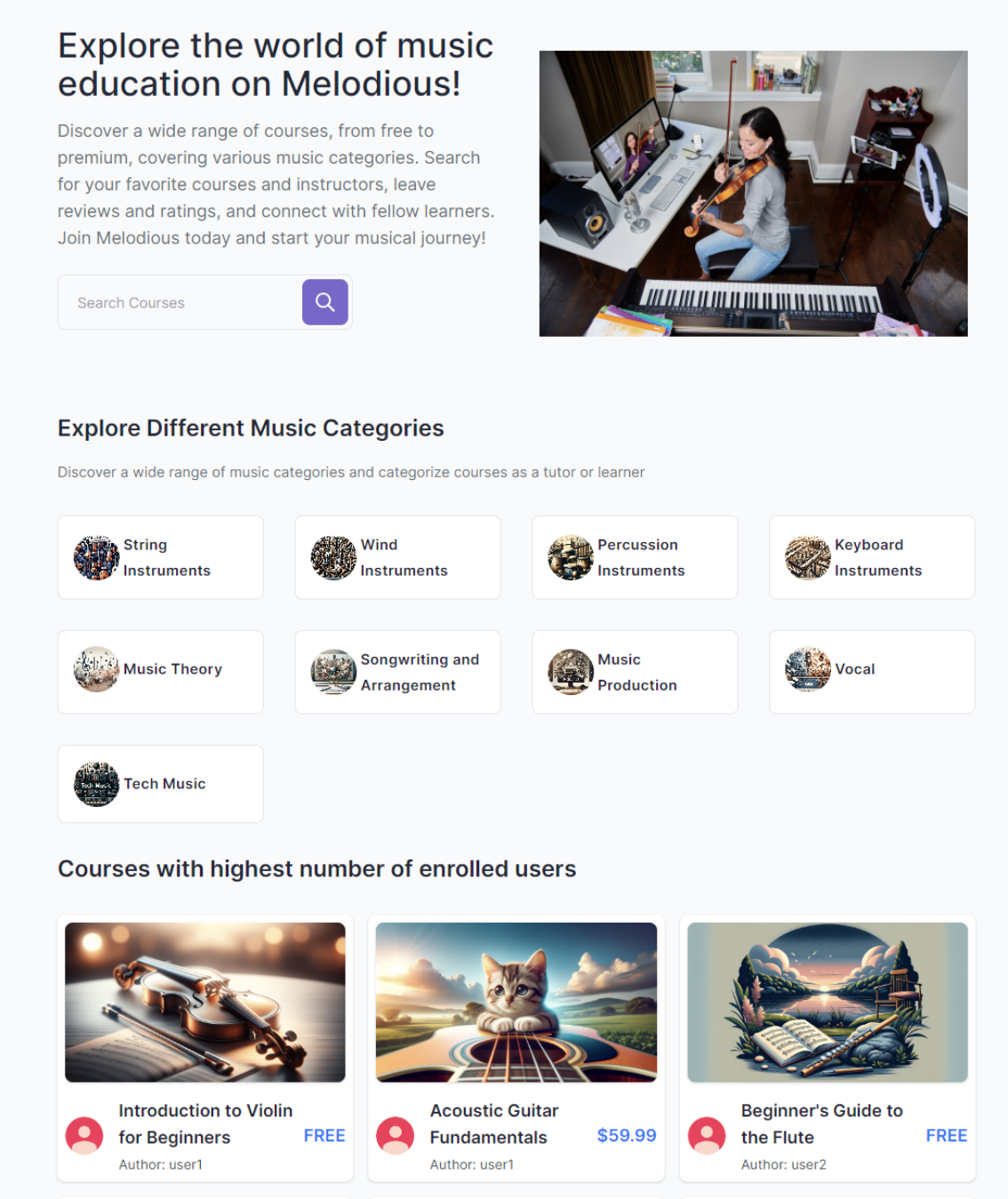


Figure 2.1: Course Browsing Section of the Homepage

With its intuitive layout and clear segmentation, the homepage efficiently acquaints new visitors with what Melodious has to offer, underlining its commitment to a user-friendly and inclusive learning environment.

## 2.2 Authorization: Login and Registration

**Registration:** The registration process on Melodious is streamlined for ease of use. New users are required to provide a *username* and a *password* to create an account. There are other optional modification a user can do in the profile settings after signing up such as add a profile picture, add "About me" section, indicate first and last names. Upon successful registration, users are directed to the Dashboard page, where they can engage with the various features of the application.

**Login:** The login page allows existing users to access their accounts by entering their username and password. Upon successful login, users are directed to the Dashboard page, where they can engage with the various features of the application.

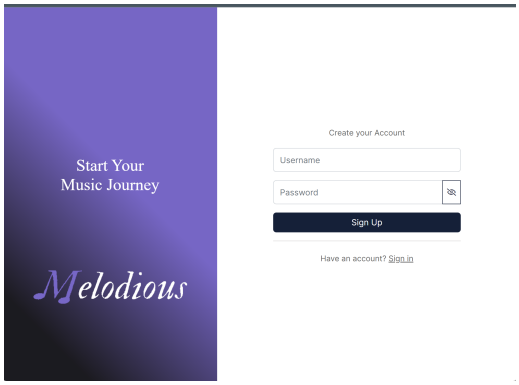


Figure 2.2: Registration page

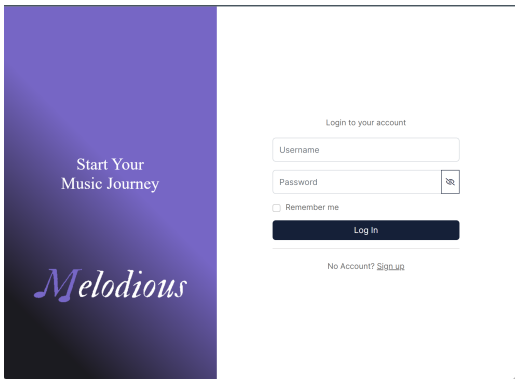


Figure 2.3: Login page

**Note:** Security is a priority for Melodious, and as such, user passwords are encrypted to enhance protection. This step ensures that personal data remains secure and private. Also the platform's server can be reached only via a special JWT tokens generated by the server itself. Further security details will be explained in Developer's Documentation.

**Future Plans:** Future updates to the authorization process may include email integration for password recovery and account verification, further enhancing user experience and security.

## 2.3 Vertical Navigation Sidebar

The Vertical Navigation Sidebar in Melodious offers direct access to the main features of the platform, each leading to its respective main page:

- **Dashboard:** A central hub for the user's activities and overview.
- **My Courses:** Management and viewing of the user's created or administered courses.
- **Explore:** Discovery and browsing of new courses available on the platform.
- **My Learning:** A list of courses the user is enrolled in or has completed.
- **Transactions:** Detailed view of financial activities, including purchases and earnings.

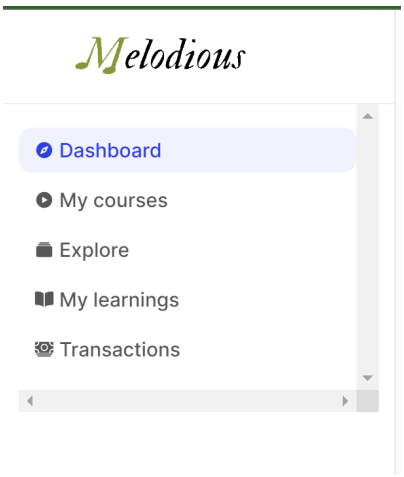


Figure 2.4: Vertical Navigation Sidebar

## 2.4 Dashboard Page Overview

The Dashboard serves as the central hub for users on Melodious, offering a comprehensive overview of their activities and interests. It is intuitively designed to provide quick access to various aspects of the platform. (Refer to Figure 2.5 below)

### 2.4.1 My Learning

This section displays all the courses the user is currently enrolled in, presented as small course cards. It offers a snapshot of the user's learning journey on Melodious.

### 2.4.2 Chats

The Chats section showcases the most recent conversations with other users, providing a preview of the last message in each chatroom. This feature facilitates easy and quick communication within the community.

### 2.4.3 Transactions

This part of the Dashboard is divided into two tabs: "My Purchases" and "My Earnings." Each tab displays a detailed table of transactions: 'My Purchases' lists courses bought by the user, while 'My Earnings' shows income from courses sold by the user.

### 2.4.4 Recommended Courses

Here, users find courses suggested specifically for them, based on their past course engagements and saved courses. The recommendation algorithm's details will be elaborated in the Developer Documentation. Recommended courses are also displayed as small course cards.

### 2.4.5 Saved Courses

In this section, users can view all the courses they have saved. Courses can be saved by clicking on a bookmark icon on the course cards, which becomes visible upon hovering over the course. Saved courses are marked for easy identification.

Each section mentioned above includes a "See All" link, providing access to a dedicated page with a detailed view for users who wish to explore more.

*The subsequent sections will detail each feature mentioned above, providing insights into the functionalities and user interaction on their respective pages.*

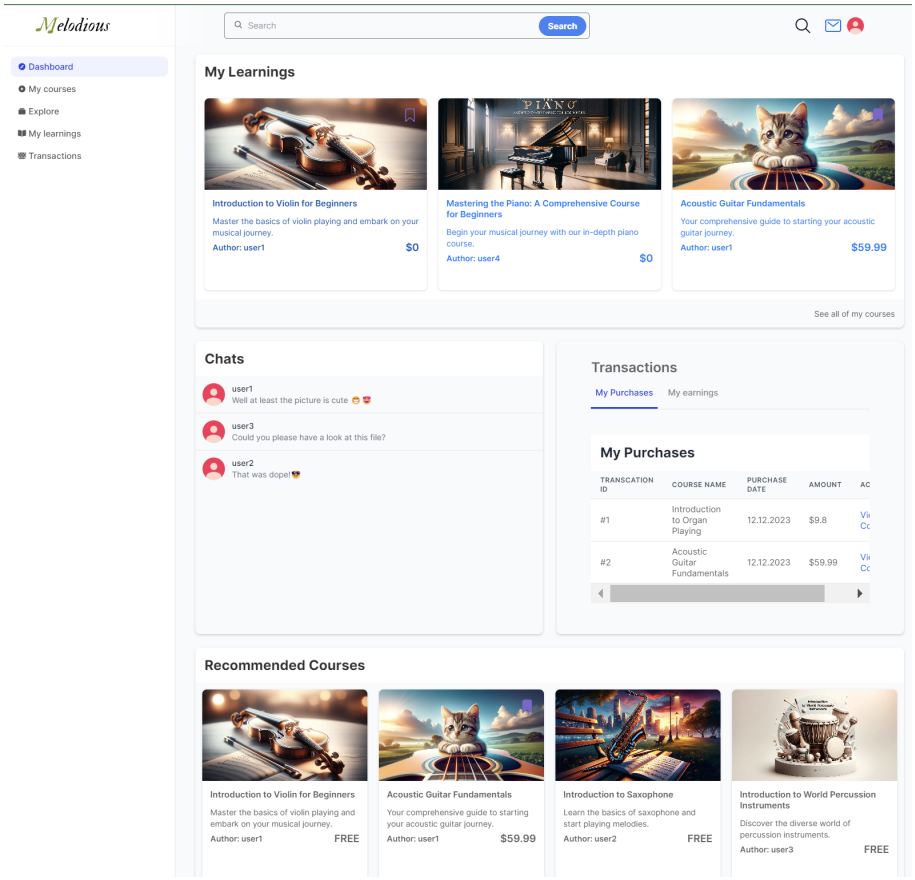


Figure 2.5: Dashboard overview

## 2.5 My Courses Page

The "My Courses" page on Melodious is designed for course authors to manage their courses. This page is divided into two primary tabs: "Published Courses" and "Draft Courses," each catering to different stages of course development.

### 2.5.1 Published Courses

The "Published Courses" tab displays a list of all courses that the user has published. These courses are visible and accessible to all users on the platform. Each course entry provides essential details and the option for the author to view the course as it appears to learners.

### 2.5.2 Draft Courses

Under the "Draft Courses" tab, users find courses that are currently in development. These courses are not visible to other users and are accessible only to the author. This section allows for editing and further development of the course material. Authors can publish these courses directly from the edit page once they are ready.

In both tabs, users are presented with a button labeled "Click here to create a new course." This button directs users to the course creation page, which will be detailed in the following section.

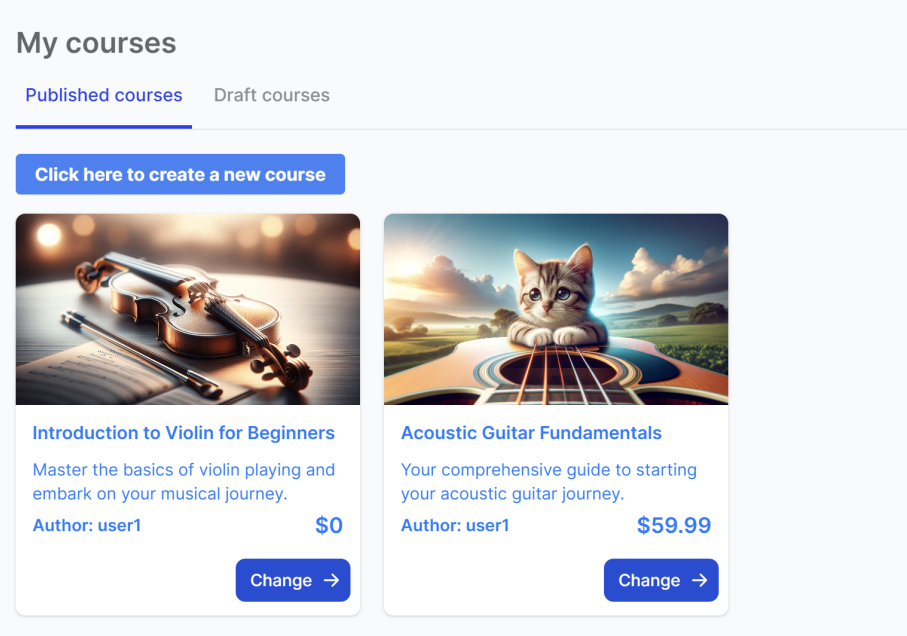


Figure 2.6: My Courses Page

## 2.6 Course Creation Page

The Course Creation Page on Melodious is a comprehensive interface for authors to create and manage their courses. This section outlines the various elements and functionalities available to the user during the course creation process.

### 2.6.1 Basic Course Information

- **Course Name:** The displayed title of the course.

- **Course Preview Picture:** A thumbnail image representing the course.
- **Course Promo Video:** A promotional video that introduces the course.
- **Short Description:** A concise summary of the course content.
- **Long Description:** A detailed description with rich text formatting options, including headers, lists, links, and text styles.
- **Course Price:** The cost of the course in dollars. A value of 0 indicates that the course is free.
- **Course Category:** A dropdown selection for the course category, with options like "String Instruments", "Wind Instruments", "Percussion Instruments", "Keyboard Instruments", "Music Theory", "Songwriting and Arrangement", "Music Production", "Vocal", "Tech Music".

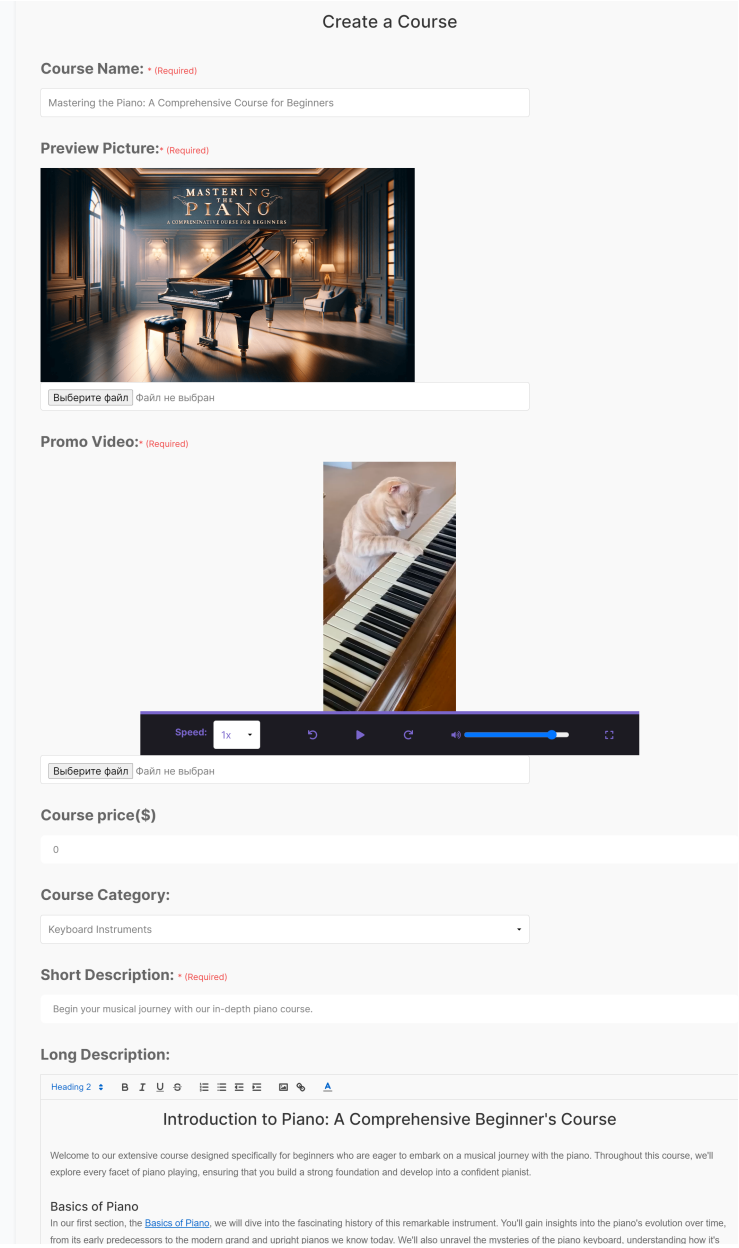


Figure 2.7: Course creation: Basic course information

### 2.6.2 Requirements Section

Authors can add multiple requirements for their course. The UI allows for easy reordering through drag-and-drop functionality and the option to add or delete requirements.

### 2.6.3 What You Will Learn Section

This section allows authors to list learning outcomes or objectives. Similar to the Requirements section, items can be added, deleted, and reordered.

### 2.6.4 Curriculum

The curriculum of a course on Melodious is organized into a hierarchical structure, starting with modules, which are further divided into topics, each with its specific content.

- **Modules:** Each module represents a major section or theme of the course. Modules serve as the overarching categories that organize the course into distinct segments.
- **Topics in Modules:** Within each module, topics represent individual subjects or concepts. Authors provide a name for each topic, defining the specific focus within the module.
- **Content in Topics:** The content of each topic is the core instructional material, which can vary in format depending on the selected content type. The options include Video, Image, Text, Document, and Quiz, each with appropriate fields and upload options.
  - **Video/Image/Doc:** Media content types, where authors upload the respective files. (See figures 2.8, 2.9, 2.10)
  - **Text:** Written content with a text field that supports rich formatting options. (See figure 2.11)
  - **Quiz:** Interactive quizzes, where authors can add questions, provide multiple-choice answers, and designate the correct options. (See figure 2.12)

This structured approach allows course authors to comprehensively cover various topics in an organized and pedagogically sound manner, enhancing the learning experience for students.

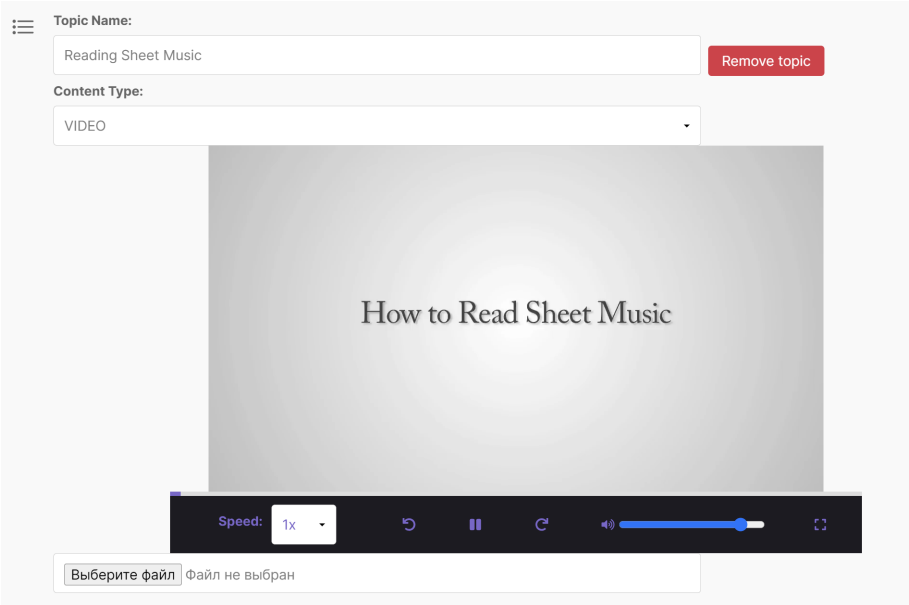


Figure 2.8: Topic creation: VIDEO content type

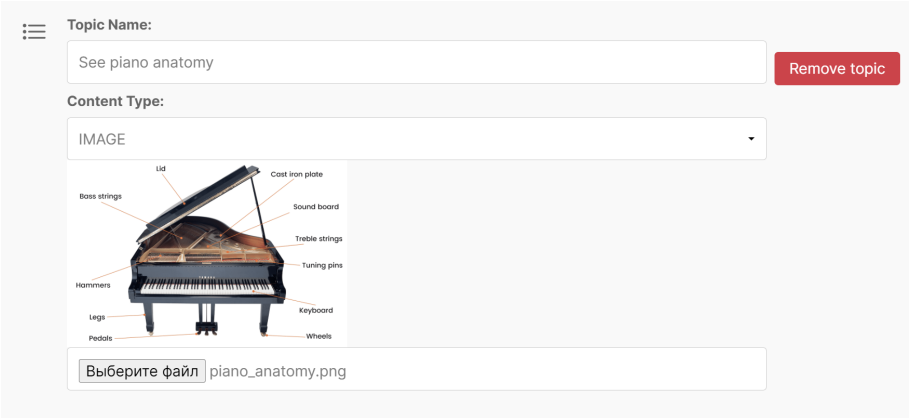


Figure 2.9: Topic creation: IMAGE content type

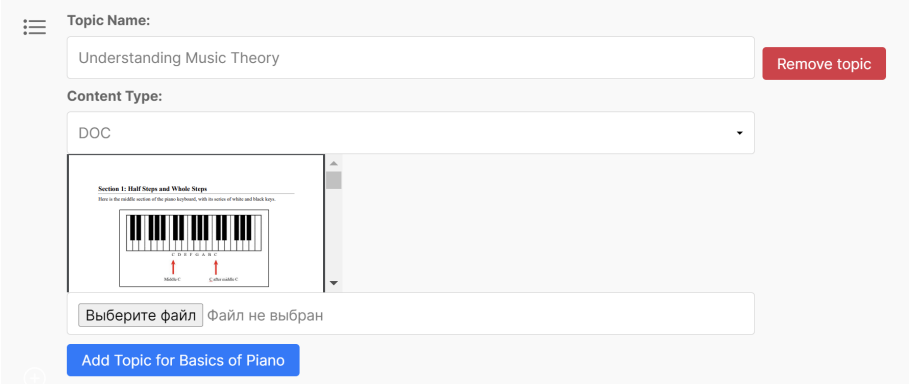


Figure 2.10: Topic creation: DOC content type

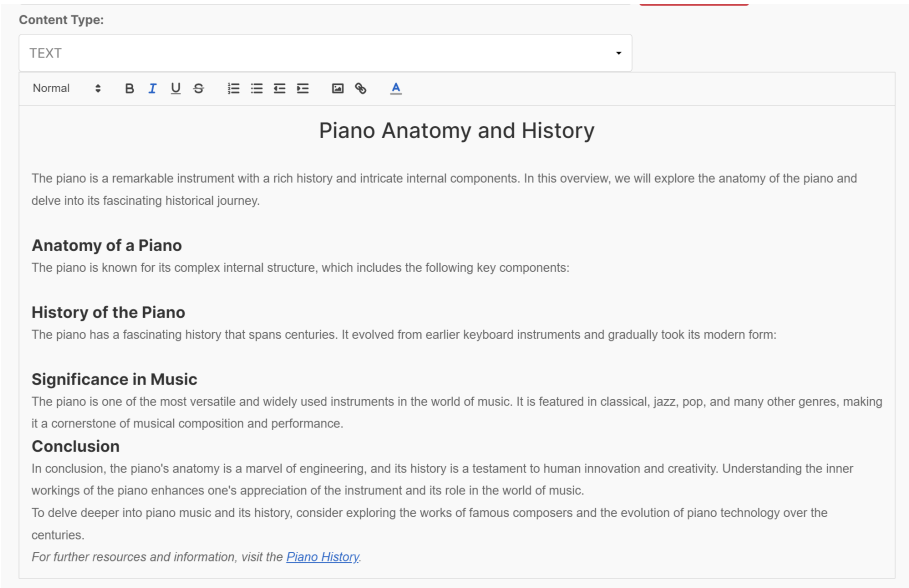


Figure 2.11: Topic creation: TEXT content type

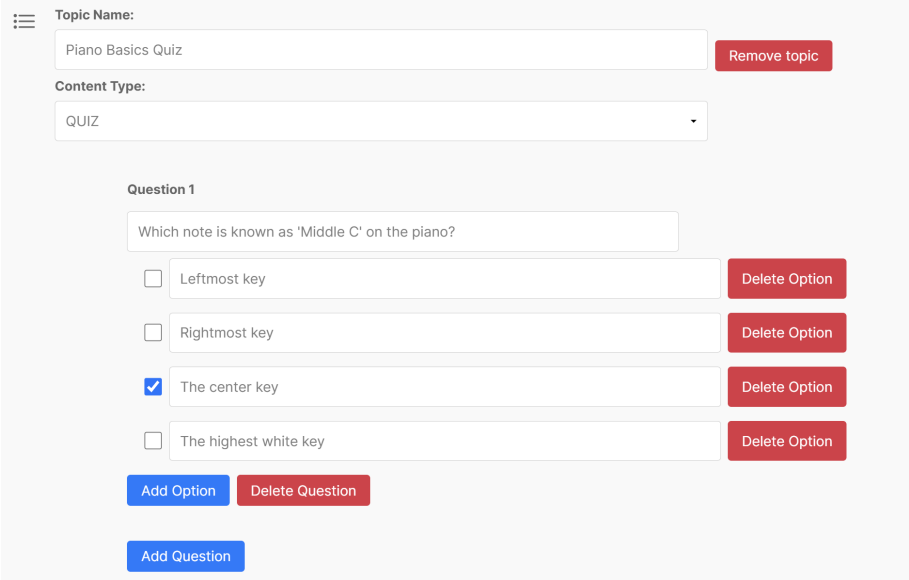


Figure 2.12: Topic creation: QUIZ content type

### 2.6.5 Tags

Up to three tags can be added to the course, enhancing searchability and discoverability on the platform.

### 2.6.6 Action Buttons

- **Save as Draft:** Saves the current state of the course on the server for later editing. Published courses can also be reverted to draft status using this option.
- **Publish this Course:** Saves and publishes the course, making it visible to users on the platform.
- **Delete this Course:** Permanently deletes the course after confirmation, with a warning that it cannot be recovered.

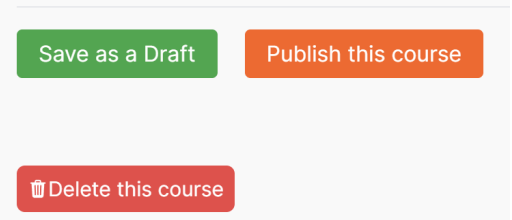


Figure 2.13: Course edit page action buttons

Such a comprehensive course creation interface empowers authors to craft detailed and engaging courses, utilizing a variety of content types and instructional materials.

## 2.7 Course Description Page

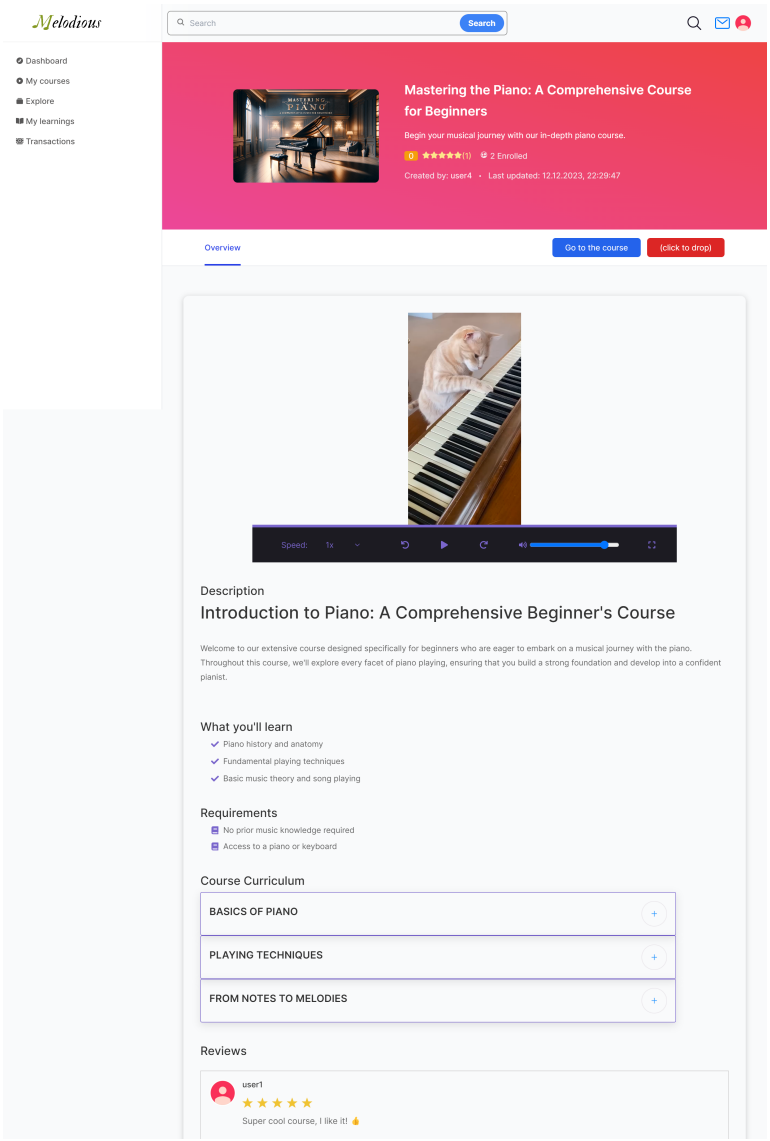


Figure 2.14: Course Description Page Overview

The Course Description Page opens up when a certain course card is clicked. It provides users with detailed information about it aiding in making informed decisions about their potential enrollment. This page is accessible for all published courses and offers insight into what the course covers, its structure, and the learning outcomes. All the information is populated from the data the author indicated during the course creation stage.

### 2.7.1 Course Overview

The top section of the page presents the course title, thumbnail image, promotional video, and a brief introduction. Here, users get an initial feel for the course content and what to expect from it.

### 2.7.2 Detailed Description

Below the overview, the page divides into sections corresponding to the course content laid out by the author during the course creation process.

- **Description:** This section expands on the course's content, providing potential learners with an in-depth look at what the course offers, including its goals and structure.
- **Modules and Topics:** The course curriculum is showcased here, with modules and their respective topics listed to give users a clear view of the course's scope and sequence.
- **Conclusion:** A final note from the course creator, often summarizing the course objectives and inviting users to enroll.

### 2.7.3 What You Will Learn

Here, users can find what they will learn from the course, typically listed in bullet points for quick and easy reference.

### 2.7.4 Requirements

This part lists the prerequisites for the course, informing users of any necessary prior knowledge or resources they may need.

### 2.7.5 Curriculum

A summarized view of the course curriculum is presented, giving users an overview of the main modules and topics within them.

### 2.7.6 Reviews

At the bottom of the page, users can read reviews from other users who have taken the course. This section provides social proof and can significantly influence a potential learner's decision to enroll.

### 2.7.7 Action Buttons

At the top of the Course Description Page, users will find action buttons that vary based on their relationship to the course.

- **For Course Authors:**
  - *Edit:* Authors will see an "Edit" button that directs them to the course creation/edit page for modifying their course. See figure 2.15
- **For Enrolled Users:**
  - *Go to Course:* This button takes enrolled users directly to the course content.
  - *Drop the Course:* Allows users to unenroll from the course, removing it from their "My Courses" list.

  See figure 2.16
- **For Potential Learners:**
  - *Enroll for Free:* If the course is free, clicking this button will immediately enroll the user and direct them to the course content. See figure 2.18
  - *Buy for x$*: For paid courses, this button redirects to a payment page, allowing the user to purchase and enroll in the course. See figure 2.17

## 2.8 Course Checkout and Payment Process

The Course Checkout Page facilitates a smooth transaction experience for users who wish to enroll in paid courses on Melodious. This process is integrated with Stripe for secure payment handling and involves a mock transaction flow for development and testing purposes.

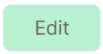


Figure 2.15: Button for the course author


Figure 2.16: Buttons for enrolled students


Figure 2.17: Button of a priced course for other users


Figure 2.18: Button of a free course for other users

### 2.8.1 Initiating the Purchase

Upon selecting to purchase a course by clicking the "Buy" button on the course description page, the user is navigated to the Course Checkout Page. Here, the course details are displayed, including the course title, author, and the price to be paid.

### 2.8.2 Completing the Transaction

The user can click the "Pay Now" button to initiate the payment process. A pop-up will appear, requesting the user to input their payment credentials:

- **Payment Information:** Users must enter their email and card details, including the card number, expiration date, and CVC. For the purposes of development, the card number "4242 4242 4242 4242" with any future expiration date and CVC can be used to simulate a successful transaction.
- **Transaction Processing:** The Stripe API, integrated with the platform's frontend and backend, processes the transaction. The mock transaction simulates a real payment without the exchange of actual funds.
- **Enrollment Confirmation:** Once the mock transaction is approved, Stripe and the platform's database record the purchase. The user is then enrolled in the course and redirected to the course content page.

### 2.8.3 Database Recording

The platform's backend records the transaction in the database, updating the user's enrollment status and granting access to the course content.

Note: Actual monetary transactions do not take place in this development environment. The process is purely for demonstration and testing purposes.

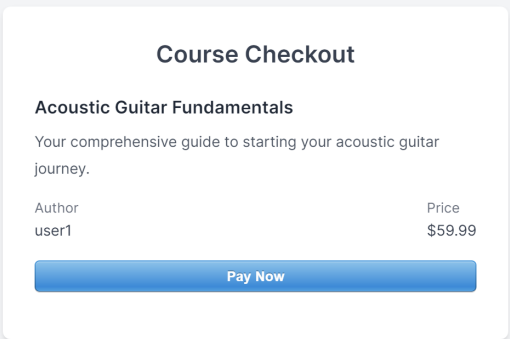


Figure 2.19: Course Checkout Page

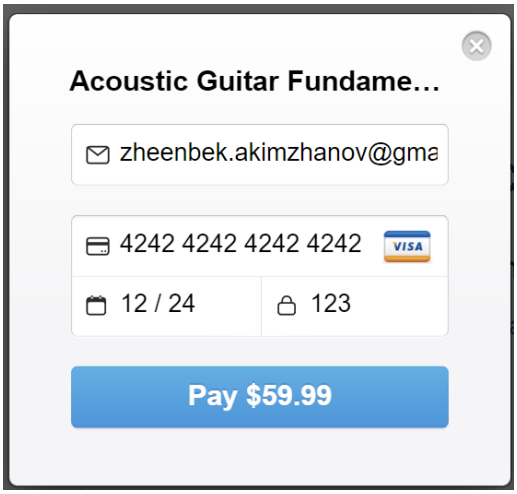


Figure 2.20: Payment Details Pop-up

## 2.9 My Learning Page

The "My Learning" page is a dedicated space for users to track and manage their course enrollments on Melodious. This page is organized into two tabs: "Taken Courses" and "Saved Courses," providing a clear and organized view of the user's learning activities.

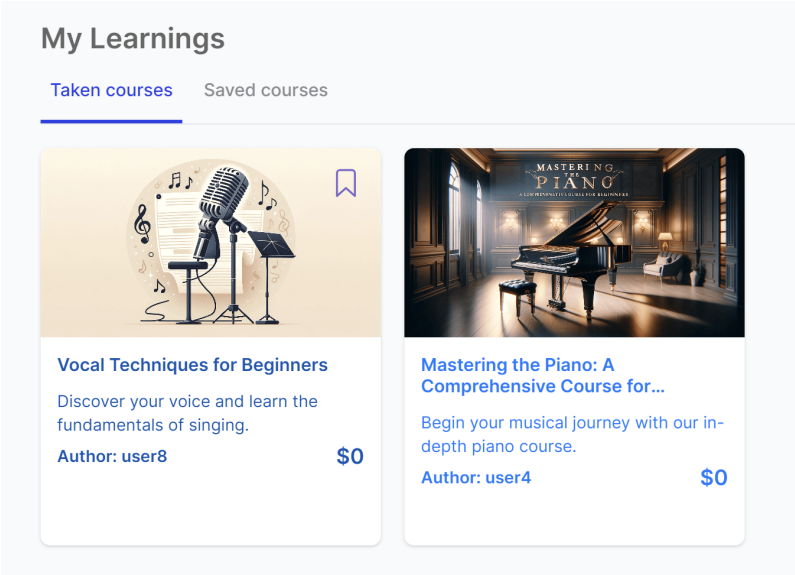


Figure 2.21: My Learning Page Overview

### 2.9.1 Taken Courses

Under the "Taken Courses" tab, users can find all the courses in which they are currently enrolled.

### 2.9.2 Saved Courses

The "Saved Courses" tab displays a collection of courses that the user has bookmarked for future reference. Bookmarking can be done from any course preview card by clicking on the bookmark icon.

### 2.9.3 Future Enhancements

Future updates to the "My Learning" page may include additional tabs such as "Completed Courses" and "Archived Courses," further enhancing the user's ability to organize their learning journey.

## 2.10 Transactions Page

The Transactions Page offers a detailed record of the financial activities related to the user's courses on Melodious. This section is accessible via the Vertical Navigation Sidebar and is divided into two tabs for clarity and organization.

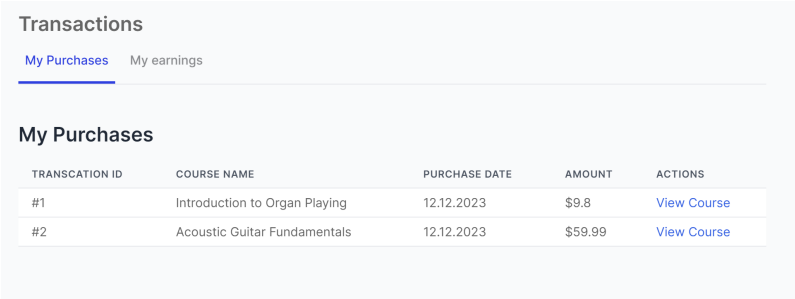


Figure 2.22: Transactions Page Overview

### 2.10.1 My Purchases

The "My Purchases" tab lists all the transactions where the current user has purchased courses as a learner. Free courses do not generate transactions. The information is displayed in a tabular format containing the following columns:

- **Transaction ID:** A unique identifier for each transaction.
- **Course Name:** The name of the course purchased.
- **Purchase Date:** The date when the course was bought.
- **Amount:** The price paid for the course.
- **Actions:** A link or button to view the course, allowing users to quickly access their purchased courses.

### 2.10.2 Your earnings

The "Your earnings" tab provides an overview of earnings from courses sold by the user. Similar to the "My Purchases" tab, this section lists the transactions made by other users who have purchased the user's courses.

## 2.11 Course Content Page

The Course Content Page is where the learning experience for a user takes place on Melodious. This page is accessible after a user enrolls in a course and contains all the educational material organized by topics.

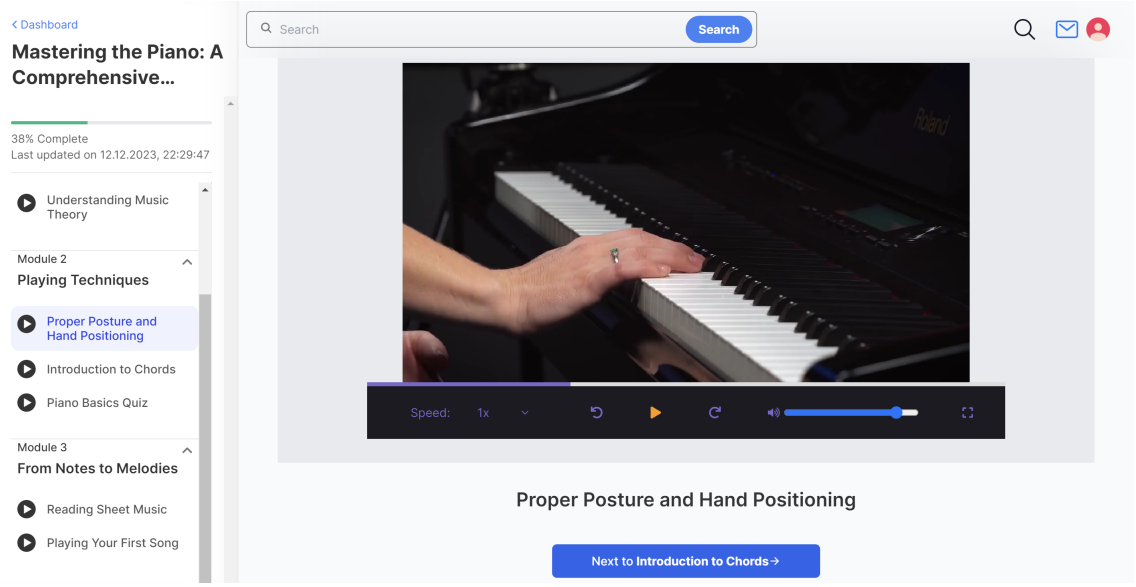


Figure 2.23: Course Content Page

### 2.11.1 Course Navigation and Progress

The left sidebar on the Course Content Page serves as the navigation panel, displaying the list of topics and the user's progress through the course. Features of this sidebar include:

- A progress bar indicating the percentage of the course completed.
- A list of modules and topics that the user can click through to navigate the course.
- Forward and backward navigation buttons to proceed to the next topic or revisit the previous one.

### 2.11.2 Topic Content Display

Each topic within a module is associated with a content type, which can be one of the following: VIDEO, IMAGE, TEXT, DOC, or QUIZ. The corresponding content is displayed in the main area of the page:

- **VIDEO:** Video lectures or demonstrations related to the topic.
- **IMAGE:** Illustrative images or infographics.
- **TEXT:** Textual information or articles.
- **DOC:** Downloadable documents such as PDFs.
- **QUIZ:** Interactive quizzes that must be completed with a minimum score of 80% to proceed.

### 2.11.3 Completion and Review

Upon finishing all topics within the course, users are prompted to leave a review. This feedback is an essential part of the learning experience on Melodious as it:

- Contributes to the course's rating.
- Is visible on the Course Description Page for prospective students to see.

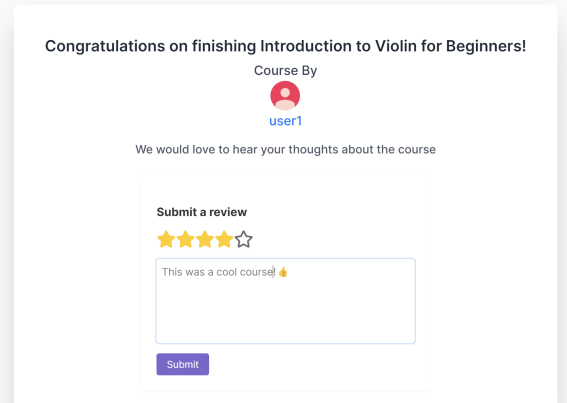


Figure 2.24: Review at course completion

## 2.12 Searching

The search feature on Melodious is an essential tool that enables users to quickly find courses and connect with other users. This robust search system allows for a comprehensive search across various attributes of courses and user profiles.

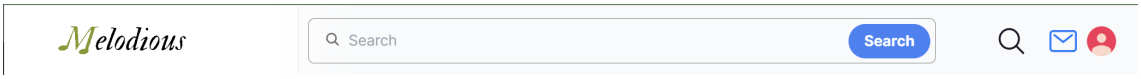


Figure 2.25: Top Navigation Bar with search, chats and profile

### 2.12.1 How to Search

Users can begin a search by simply typing into the search bar located at the top of every page and pressing enter to start the query.

### 2.12.2 Search Results

The search results are categorized under two tabs for ease of navigation:

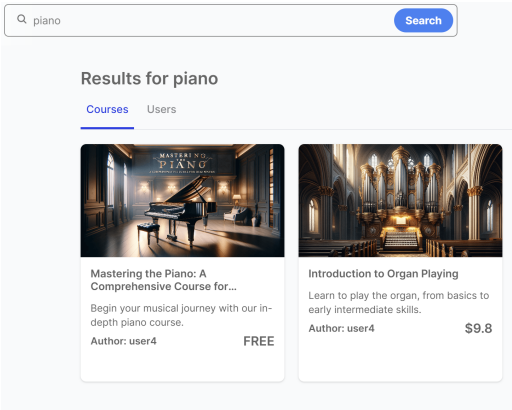


Figure 2.26: Search results: courses

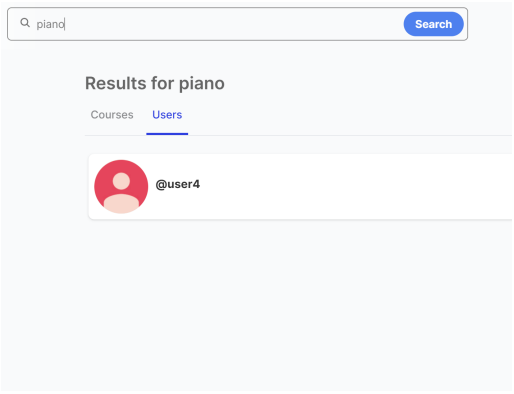


Figure 2.27: Search results: users

- **Courses:** This tab displays courses that are relevant to the search query. The algorithm checks the course name, descriptions, tags and the author's information to ensure comprehensive results. Only published courses are included in the search results, ensuring that users only see courses that are currently available to them.
- **Users:** The users' tab lists members of Melodious whose profiles or published courses contain the search keywords. This allows users to find and connect with instructors or other learners who share similar interests or expertise.

## 2.13 Profile Page

The Profile Page is a personalized space on Melodious that showcases all the relevant information about a user. This comprehensive section allows users to learn more about each other, fostering a community of learning and sharing.

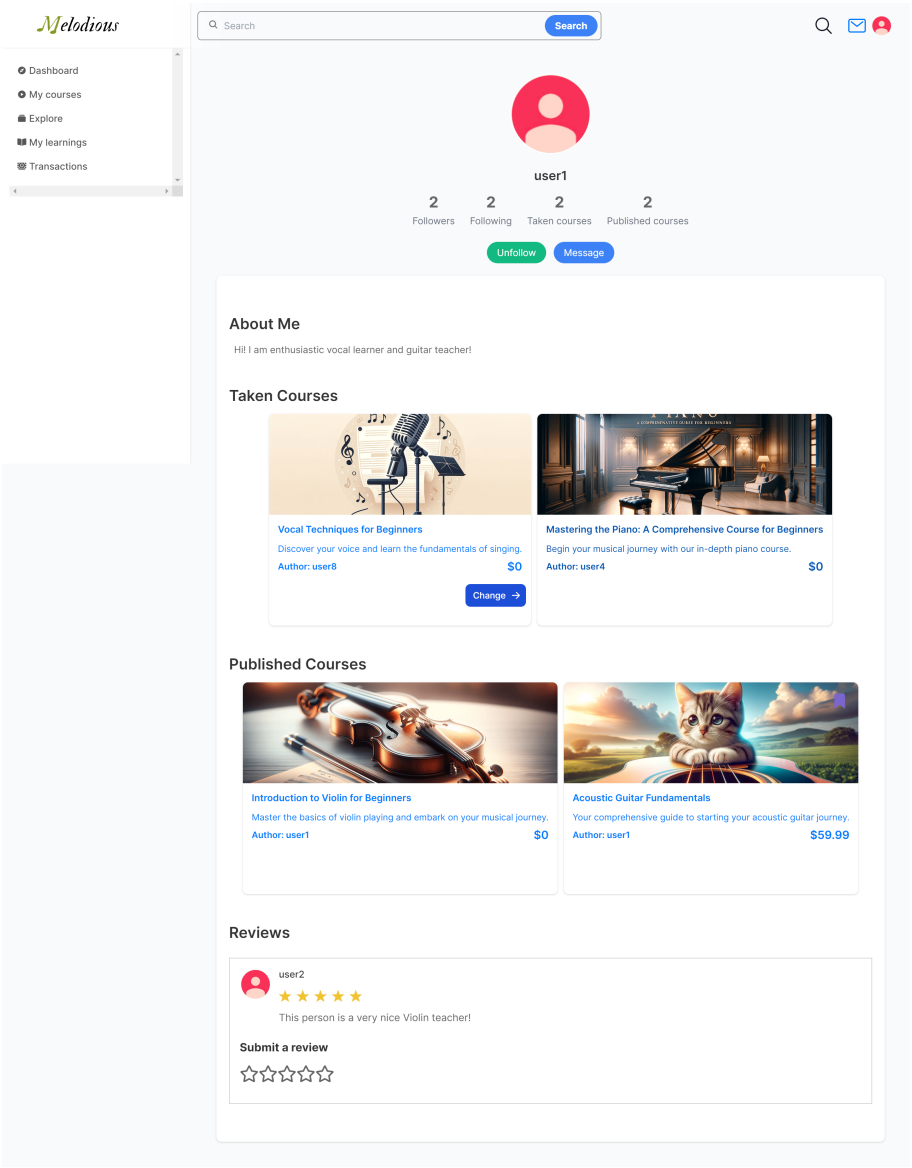


Figure 2.28: Profile Page Overview

### 2.13.1 Accessing Profiles

Users can navigate to any user's profile page by clicking on usernames located in course details or reviews, or by utilizing the search function detailed in Section 2.12.

### 2.13.2 Profile Overview

At the top of the profile page, you will find the user's profile picture and a quick stats section displaying:

- The number of followers and followings
- The total number of courses the user has taken
- The number of courses the user has published

Users have the option to follow or unfollow others and to send direct messages, facilitating interaction and communication.

### 2.13.3 About Me

This section provides a personal bio of the user, giving insights into their interests, expertise, and background in music learning and teaching.

### 2.13.4 Taken Courses

Here, users can see the courses that the profile owner has enrolled in. It reflects the user's learning path and interests, which can be an inspiration for others.

### 2.13.5 Published Courses

The Published Courses section lists all the courses that the user has created and made available on Melodious. This serves as a portfolio of the user's contributions to the platform.

### 2.13.6 Reviews

In the Reviews section, users can read feedback and ratings from:

- Other users who have taken the profile owner's courses
- Authors of courses the profile owner has taken.

### 2.13.7 Review Submission

Users visiting a profile can submit reviews based on their experiences with the user's courses, whether as a learner or a teacher. This feature contributes to the trustworthiness and community-driven quality assurance of Melodious.

*Note: For privacy reasons in the future there will be option to hide some parts of the information in user's settings.*

## 2.14 Account Settings

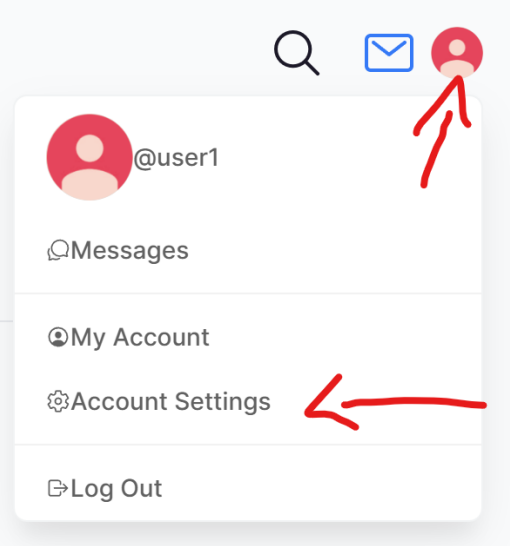


Figure 2.29: Path to settings

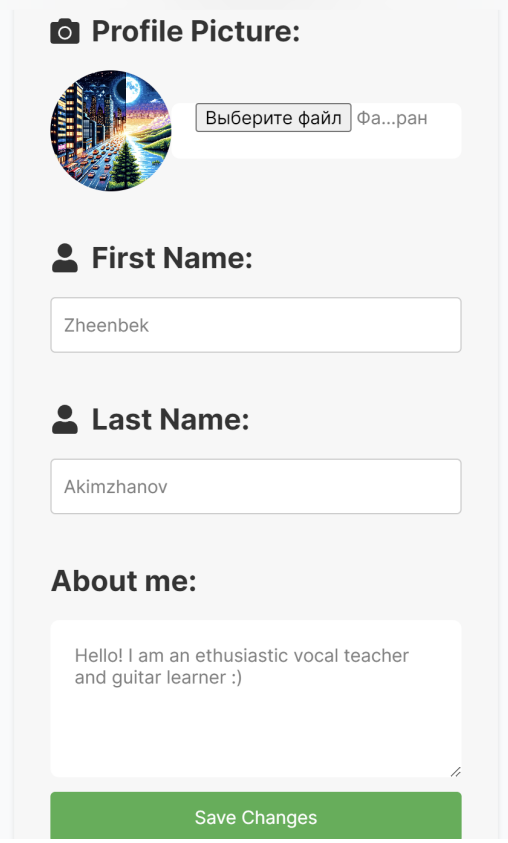


Figure 2.30: Settings page

Account Settings is accessible via the top navigation bar, allowing users to personalize their profile and manage account details:

### 2.14.1 Personalization

Users can upload a profile picture and provide their first and last names, which are displayed on their public profile page. Additionally, the 'About Me' section lets users share more about themselves with the Melodious community.

**Note:** Future updates will introduce additional enhancements for account settings include adding email for account recovery, changing the username and pass-

word, and more, ensuring users have full control over their account security and public persona.

## 2.15 Messages Page

The Messages Page on Melodious serves as a hub for user communication, following the standard messaging apps interface to provide a familiar and intuitive messaging experience.

### 2.15.1 Accessing Messages

To access the messaging interface, users can click on the envelope icon located in the top navigation bar. This action reveals the Messages Page, which is split into two main sections:

- **Chat List:** Displayed on the left, this section lists all existing chats, showing the most recent message as a preview for each chat. Users can select any chat to view the full conversation.
- **Chat Display:** On the right, the selected chat is displayed, where users can read the conversation history and continue messaging.

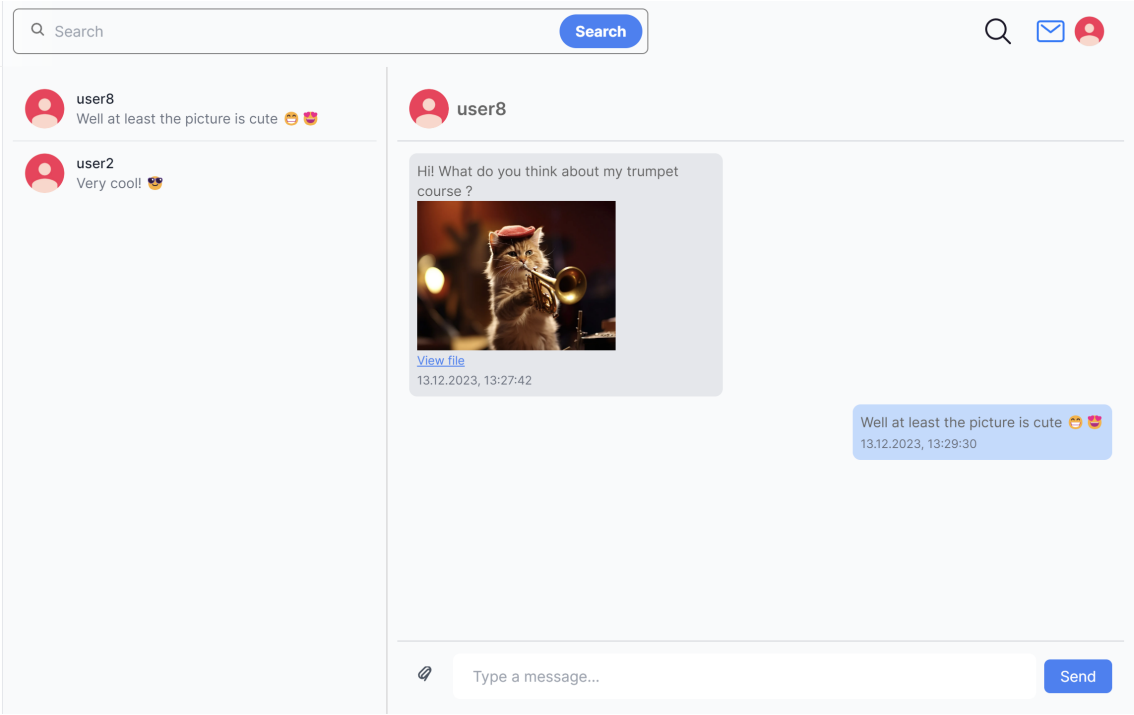


Figure 2.31: Messages Page Interface

### 2.15.2 Starting New Chats

To initiate a chat with a new user, the following steps are taken:

1. Utilize the search bar to find the user by username.
2. Navigate to the user's profile page by selecting their name from the "Users" tab in the search results.
3. Click the "Message" button on the user's profile page to open a new chatroom.

Once a message is sent, the chatroom will appear in the chat list for easy future access.

### 2.15.3 Chat Functionality

Within each chat, users have the capability to:

- Send text messages.
- Attach and send various file types, including video files, pictures, and documents.

The order of chats in the chat list is dynamic, updating based on the most recent message sent or received, ensuring that active conversations are promptly accessible.

# Chapter 3

# Developer Documentation

This chapter presents a detailed overview of the technical aspects of Melodious, a music education platform developed with ReactJS, Java Spring Boot, and MySQL. It covers the architectural design, system components, and development processes, providing insights into the frontend and backend implementation, database design, and security measures. The aim is to offer a clear understanding of the platform's technical framework and development rationale, serving as a guide for future development and maintenance. This documentation is intended for developers and technical personnel involved in supporting, enhancing, or understanding the Melodious platform's inner workings.

## 3.1 Technology Overview and Setup Guidance

This section outlines the step-by-step process for setting up the Melodious platform, including installations for both Windows and Linux systems.

### 3.1.1 Prerequisites

This section provides an overview of the key technologies used in developing Melodious, alongside detailed instructions for their installation and setup. The focus is on technologies essential for running the application, with additional tools used during development briefly discussed at the end.

- MySQL Database

- Git
- Java JDK (Version 11)
- Maven (for Java project management)
- Node.js and npm (for the React frontend)

*Note:* If you already have these installed, verify they are the correct versions.

### 3.1.2 MySQL Database

**Rationale:** MySQL[1] was selected for its reliability and performance as an open-source relational database management system. It provides a robust platform for handling Melodious' data storage and retrieval needs.

**Installation in Windows:**

1. Download the MySQL Installer from the official MySQL website and follow the installation wizard.
2. Set the root password as 'rootuser' during installation.

**Installation in Linux:**

```
sudo apt-get update
sudo apt-get install mysql-server
sudo mysql_secure_installation
```

Set the root password as 'rootuser' when prompted.

**Creating the Database:** Open MySQL Shell and execute:

```
CREATE DATABASE studentdb;
```

### 3.1.3 Git Installation and Repository Cloning

**Rationale:** Git is essential for version control, allowing for efficient tracking and management of code changes throughout the development process.

**Windows and Linux installations:** Install Git from the Git website. Clone the repository:

```
git clone https://github.com/johnnyCake1/musicEducationPlatform
```

Navigate to the project root directory.

### 3.1.4 Java Spring Boot (Backend)

**Rationale:** Java Spring Boot[2] offers a powerful, flexible framework for building web applications. It simplifies backend development with its convention-over-configuration approach, robust dependency injection, and a wide range of starters.

**Windows and Linux Installation:** Check your Java version with 'java -version'. If not version 11, download JDK 11 from the Oracle website.

#### Maven (Java Project Management)

**Rationale:** Maven is used for its comprehensive project management capabilities, including dependency management and build automation, which streamline the Java development process.

**Windows and Linux Installation:** Install Maven if not already installed, following instructions from the Maven website.

#### Running the Java Backend

Ensure 'application.properties' matches your setup. Start the Java application:

```
mvn clean install
mvn spring-boot:run
```

### 3.1.5 Node.js and npm Setup for React Frontend

**Rationale:** Node.js and npm are crucial for managing dependencies and running the ReactJS application. Node.js offers a JavaScript runtime, while npm is an efficient package manager.

**Windows and Linux:** Check Node.js and npm versions with 'node -v' and 'npm -v'. If not installed or outdated, download from the Node.js website. My node version: "v20.9.0" and npm version "10.1.0".

#### Running the Frontend

Navigate to the 'ui' directory and install dependencies:

```
cd ui
npm install
```

Ensure 'constants.js' has the correct API URL. Start the frontend:

```
npm start
```

Ensure that both frontend and backend are running correctly and can communicate with each other.

### 3.1.6 Additional Development Tools

**IntelliJ and Microsoft Visual Studio:** Used for their comprehensive IDE capabilities in Java and frontend development, respectively. **Postman:** Employed for testing and interacting with the API. **Stripe Payment API[3] :** Integrated for managing secure online transactions within the platform.

*Note:* The additional development tools are not mandatory for running the application but were instrumental in the development process.

## 3.2 Problem Specification

As highlighted in the introduction 1, Melodious addresses the fragmentation in online music education by creating a unified, community-driven platform for music teachers and learners. It responds to the need for a centralized resource where users can both teach and learn, seamlessly switching roles. The platform distinguishes itself with a user-friendly interface, mirroring social media dynamics but centered around music education. Emphasizing communication and social collaboration, Melodious aims to evolve into a central hub for the global music community, facilitating shared knowledge and growth in an interactive, dynamic environment.

## 3.3 System Design and Architecture

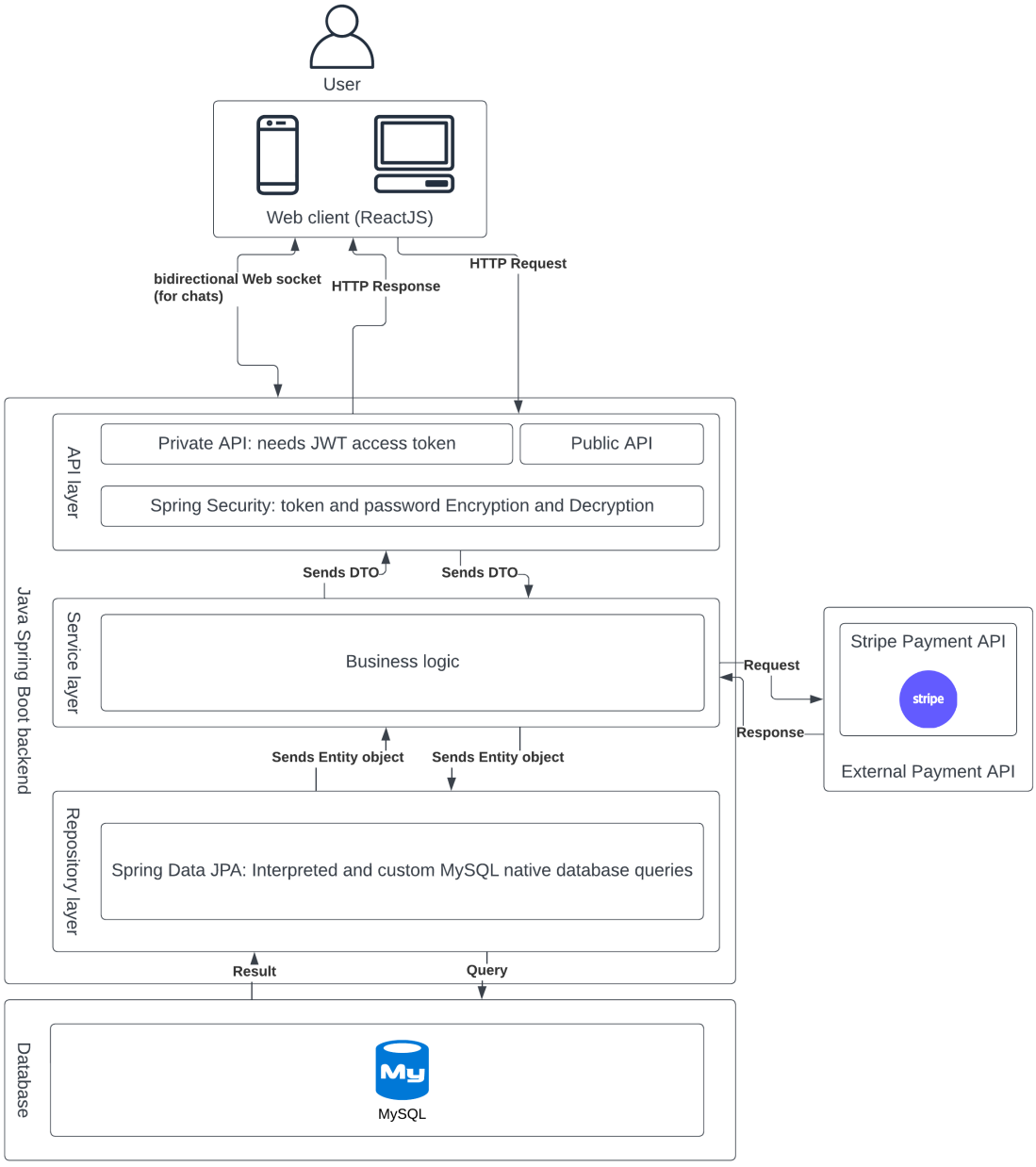


Figure 3.1: System Architecture Diagram

This section delves into the structural design of Melodious, detailing its software architecture, the interaction between frontend and backend components, and the database schema design.

### 3.3.1 Software Architecture

Melodious employs a three-tier architecture, consisting of the client-side frontend, the server-side backend, and the database layer. This modular approach ensures separation of concerns, making the system more manageable and scalable (See figure 3.1). Key components include:

- **Frontend:** Built with ReactJS, the frontend is designed as a single-page application (SPA) that communicates with the backend through RESTful APIs and employs bidirectional web sockets for real-time chat functionalities.
- **Backend:** The backend, developed with Java Spring Boot, handles business logic, API requests, and database interactions. It is structured into various modules such as user authentication, course management, and payment processing. Spring Security is used to manage security concerns such as authentication, authorization, and secure communication.
- **Data Layer:** MySQL is used for data persistence, storing user data, course content, and transaction records.
- **External APIs:** Integration with external APIs, like Stripe for payment processing, extends the platform's functionality.

### 3.3.2 Frontend and Backend Interaction

The interaction between frontend and backend is primarily managed through RESTful API calls (see HTTP calls shown in the fugure 3.1). JSON is used as the data exchange format, offering a lightweight and language-independent medium. The process typically involves:

- The frontend sends HTTP requests (GET, POST, PUT, DELETE) to the backend.
- The backend processes these requests, interacts with the database if needed, and returns a response.
- The frontend then parses this response and updates the UI accordingly.

Token-based authentication (JWT) is employed for secure communication, ensuring that API calls are validated and authorized.

## 3.4 API Documentation

This section outlines the RESTful API endpoints provided by the Melodious platform, detailing their HTTP methods, URI patterns, request parameters, response structures, and the authentication and authorization mechanisms in place.

### 3.4.1 RESTful API Endpoints

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/auth/register | Endpoint to initiate user registration. Returns a message for taken username. |
| POST | /api/v1/auth/register | Creates a new user account with the provided credentials. |
| **Example JSON Object:**<br>{<br>"username": "newuser",<br>"password": "password123"<br>} | | |
| POST | /api/v1/auth/login | Authenticates a user and returns a JWT token upon successful login. |
| **Example JSON Object:**<br>{<br>"username": "existinguser",<br>"password": "password123"<br>} | | |
| GET | /api/v1/auth/validate | Validates the provided JWT token for user authentication. No JSON object required, token passed in header. |

Table 3.1: API Endpoints for Authentication

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/courses/recommendations | Retrieves course recommendations for a user based on the given user ID. Requires authorization. |
| POST | /api/v1/courses | Creates a new course with provided details and media files. Requires authorization. |
| **Example JSON Object:**<br>{<br>"authorId": 1,<br>"courseName": "Music Theory Basics",<br>"price": 49.99,<br>...<br>} | | |
| PUT | /api/v1/courses | Updates an existing course or creates a draft based on the provided course information. Requires authorization. |
| **Example JSON Object:**<br>{<br>"id": 2,<br>"courseName": "Advanced Music Theory",<br>"isPublished": false,<br>...<br>} | | |
| GET | /api/v1/courses | Retrieves all published courses. Can be accessed without authorization. |

Table 3.2: API Endpoints for Courses - Part 1

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/courses/filter | Retrieves courses based on filtering criteria like most enrollments. Can be accessed without authorization. |
| GET | /api/v1/courses/{id} | Retrieves details of a specific course by course ID. Can be accessed without authorization. |
| DELETE | /api/v1/courses/{id} | Deletes a specific course by course ID. Requires authorization. |
| POST | /api/v1/courses/{id}/convert-to-draft | Converts a published course to a draft by course ID. Requires authorization. |
| DELETE | /api/v1/courses/{id}/{courseModuleId} | Deletes a specific module from a course by course and module ID. Requires authorization. |
| GET | /api/v1/courses/{id}/reviews | Retrieves reviews for a specific course by course ID. Can be accessed without authorization. |
| POST | /api/v1/courses/{id}/reviews | Adds a review to a specific course by course ID. Requires authorization. |

Table 3.3: API Endpoints for Courses - Part 2

| Method | Endpoint | Description |
|---|---|---|
| **Example JSON Object:**<br>{<br>"rating": 5,<br>"reviewMessage": "Great course!",<br>"reviewer": { "id": 3 }<br>} | | |
| DELETE | /api/v1/courses/{id}<br>/{courseModuleId}/{topicId} | Deletes a specific topic from a course module by course, module, and topic ID. Requires authorization. |
| POST | /api/v1/courses/{id}/enroll | Enrolls a user in a course by course ID. Requires authorization. |
| **Example JSON Object:**<br>{<br>"userId": 4,<br>"token": "stripe-token",<br>"amount": 50.0<br>} | | |
| POST | /api/v1/courses/{id}/drop | Drops a user from a course by course ID. Requires authorization. |
| GET | /api/v1/courses/categories | Retrieves all course categories. Can be accessed without authorization. |
| GET | /api/v1/courses/categories/{categoryId} | Retrieves a specific category by category ID. Can be accessed without authorization. |

Table 3.4: API Endpoints for Courses Part - 3

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/courses/categories /{categoryId}/get-courses | Retrieves courses by category ID. Can be accessed without authorization. |
| GET | /api/v1/courses/draft-courses | Retrieves all draft courses by author ID. Requires authorization. |
| POST | /api/v1/course/draft-courses /create-empty | Creates an empty draft course for a user. Requires authorization. |
| POST | /api/v1/courses/draft-courses /{draftCourseId}/create-empty-module | Creates an empty module in a draft course. Requires authorization. |
| POST | /api/v1/courses/draft-courses /{draftCourseId}/{moduleId}/create-empty-topic | Creates an empty topic in a course module. Requires authorization. |
| GET | /api/v1/courses/search/{keyword} | Searches all courses by a keyword. Can be accessed without authorization. |

Table 3.5: API Endpoints for Courses Part - 4

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/users | Retrieves all users. Requires authorization. |
| GET | /api/v1/users/{userId} | Retrieves a specific user by user ID. Requires authorization. |
| PUT | /api/v1/users/{userId} | Updates user information for a specific user ID. Requires authorization. |
| **Example JSON Object:**<br>{<br>"username": "updatedUser",<br>"firstName": "John",<br>"lastName": "Doe",<br>...<br>} | | |
| GET | /api/v1/users/{userId}/profile-picture | Retrieves the profile picture of a specific user. Requires authorization. |
| DELETE | /api/v1/users/{userId}/profile-picture | Deletes the profile picture for a specific user. Requires authorization. |

Table 3.6: API Endpoints for User Management - Part 1

| | | |
|---|---|---|
| POST | /api/v1/users/{userId}/follow | Follows another user based on the provided user ID. Requires authorization. |
| POST | /api/v1/users/{userId}/unfollow | Unfollows another user based on the provided user ID. Requires authorization. |
| GET | /api/v1/users/{userId}/saved-courses | Retrieves all saved courses for a specific user. Requires authorization. |
| POST | /api/v1/users/{userId}/saved-courses/{courseId} | Adds a course to the saved courses of a specific user. Requires authorization. |
| DELETE | /api/v1/users/{userId}/saved-courses/{courseId} | Removes a course from the saved courses of a specific user. Requires authorization. |
| GET | /api/v1/users/{userId}/taken-courses | Retrieves all courses taken by a specific user. Requires authorization. |
| GET | /api/v1/users/{userId}/published-courses | Retrieves all courses published by a specific user. Requires authorization. |

Table 3.7: API Endpoints for User Management - Part 2

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/purchase-records/buyer/{buyerId} | Retrieves purchase history for a specific buyer by their user ID. Requires authorization. |
| GET | /api/v1/purchase-records/seller/{sellerId} | Retrieves purchase history associated with a specific seller user ID. Requires authorization. |
| GET | /api/v1/purchase-records/course/{courseId} | Retrieves purchase history for a specific course by course ID. Requires authorization. |

Table 3.8: API Endpoints for Purchase Records

| Method | Endpoint | Description |
|---|---|---|
| GET | /api/v1/messages/{senderId} /{recipientId} | Retrieves chat messages between two users, identified by sender and recipient IDs. Requires authorization. |
| GET | /api/v1/messages/{userId} | Retrieves chat rooms of a specific user by user ID. Requires authorization. |

Table 3.9: API Endpoints for Chat

| Method | Endpoint | Description |
|---|---|---|
| POST | /api/v1/files | Creates a new file. Requires authorization. |
| POST | /api/v1/files/user-storage/{userId}/{fileId} | Saves a file for a specific user, identified by user and file IDs. Requires authorization. |
| DELETE | /api/v1/files/user-storage/{userId}/{fileId} | Removes a file from a user's storage, identified by user and file IDs. Requires authorization. |
| GET | /api/v1/files/{fileId} | Retrieves a specific file by file ID. Requires authorization. |

Table 3.10: API Endpoints for File Management

### 3.4.2 Authentication and Authorization

The Melodious platform incorporates JWT (JSON Web Token) for securing its API endpoints. This approach ensures that users are authenticated and authorized appropriately before they can access specific functionalities of the platform. The authentication process begins with the user sending their credentials to the '/api/v1/auth/login' endpoint. Upon successful authentication, the server issues a JWT, which encapsulates the user's identity and privileges.

For each subsequent request to protected endpoints, the client must include this token in the HTTP Authorization header. The server then decodes and verifies the token to confirm the user's identity and authorization level. This mechanism provides a secure and efficient way to manage user sessions and access control within the Melodious platform.

```
GET /api/v1/courses HTTP/1.1
Host: api.melodious.com
Authorization: Bearer jwt.token.here
```

## 3.5 Frontend Development

### 3.5.1 ReactJS Components Structure and Navigation

The Melodious application is structured using ReactJS[4], with a focus on modular and dynamic user interfaces. The core aspects of this structure include:

- **React Routes for Navigation**: The application utilizes React Router for navigating between different pages. Each route in the ‘App.js‘ file corresponds to a specific component, rendering different pages based on the URL path.
- **Private Route Handling**: The ‘PrivateRoute.js‘ component is crucial for access control. It wraps around components that require user authentication, checking for a valid session before granting access. If the user is not authenticated, it redirects them to the login page.
- **Session Management with Local Storage**: Authentication pages like ‘Register.js‘ and ‘Login.js‘ utilize the ‘useLocalStorageState‘ service to store and retrieve session information (‘currentUser‘ and ‘jwt‘) in the browser’s local storage. This approach ensures persistence of user sessions across different components of the application.
- **Component Structure Hierarchy**: The project is organized within the "ui" folder, encompassing a variety of subdirectories for different functionalities:
  - The *Public* folder contains static assets like SVGs, PNGs, and HTML files.
  - The *Source (src)* folder includes:
    - *App Files*: Main app component files like ‘App.css‘ and ‘App.js‘.
    - *Components*: This directory contains various subdirectories for different pages (Authentication, Chat, Course, Dashboard, etc.), each with their own stylesheets and components.
    - *Services and Utilities*: For backend communication and utility functions, files like ‘httpReqAsync.js‘, ‘webSocketService.js‘, and ‘useLocalStorageState.js‘ are used.

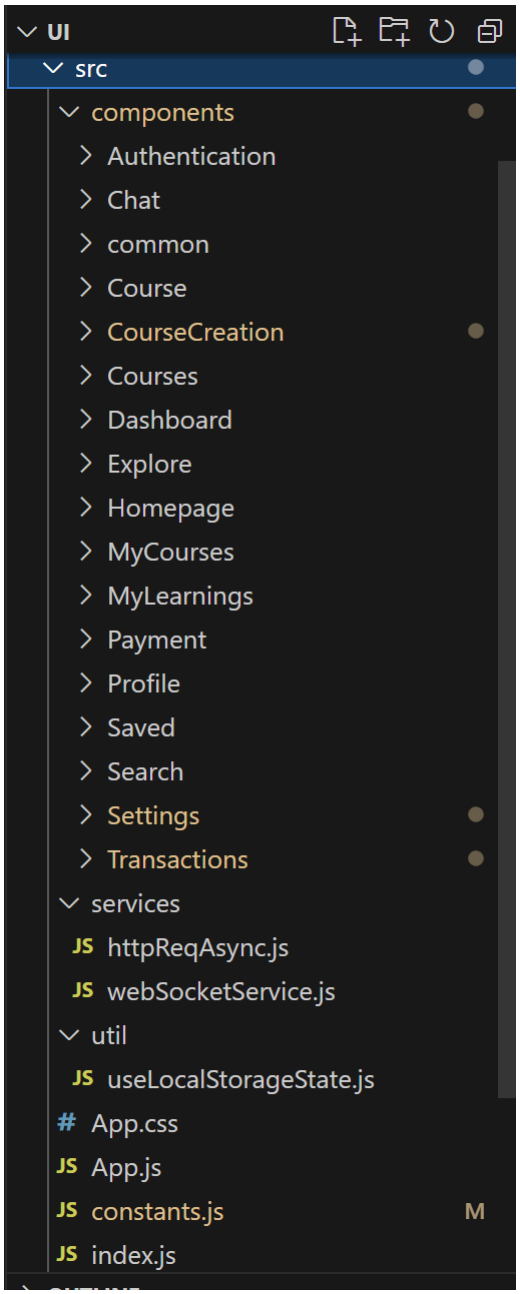


Figure 3.2: React UI folder structure

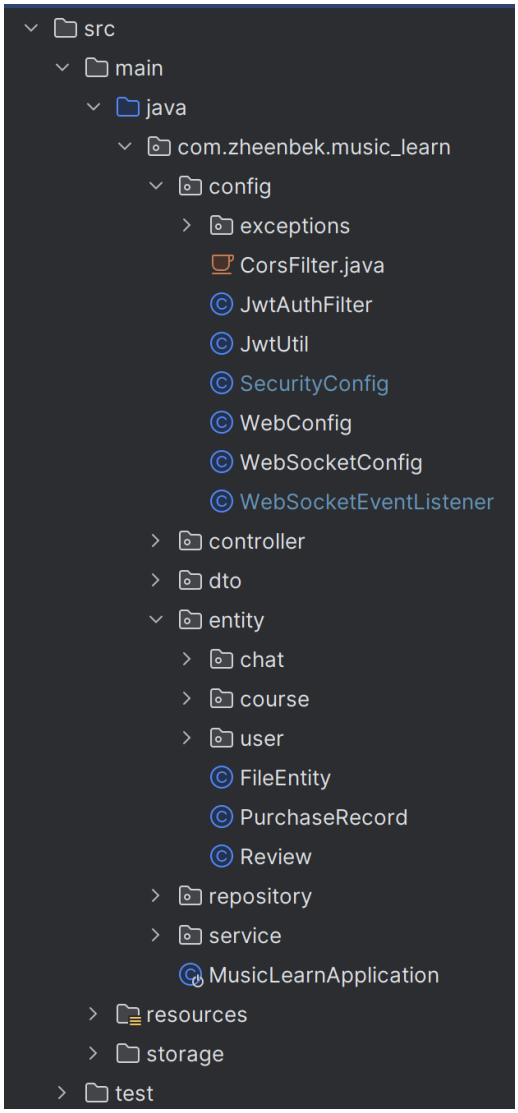


Figure 3.3: Java backend folder structure

### Example: Dashboard Backend Interaction and State Management

The ‘Dashboard.js‘ component showcases the interaction with the backend and state management within the Melodious application. This component uses the ‘httpReqAsync‘ service to perform API calls, retrieving data like the user’s courses, saved items, and chat rooms. The ‘httpReqAsync‘ service efficiently handles both successful responses and errors from the API.

```
useEffect(() => {
```

```
 if (jwt && currentUser) {
  httpReqAsync(‘/api/v1/users/${currentUser.id}/taken-courses‘,
               ’GET’,
                jwt)
  .then((result) => {
    setTakenCourses(result);
   });
 }
}, [jwt, currentUser]);
```

State management in ‘Dashboard.js‘ is handled using React’s ‘useState‘ along with ‘useEffect‘ hook. This hook is used to perform side effects in the component, such as fetching data when the component mounts or when certain dependencies change. In this example, ‘useEffect‘ is triggered when either ‘jwt‘ or ‘currentUser‘ changes. The ‘httpReqAsync‘ call within the ‘useEffect‘ hook fetches the user’s taken courses and updates the component’s state with this data using the ‘setTakenCourses‘ state setter function.

This approach ensures that the component re-renders with the latest data whenever the user’s token or profile changes, keeping the user interface up-to-date with the backend data.

```
const Dashboard = () => {
  const [takenCourses, setTakenCourses] = useState([]);
  const [recommendedCourses, setRecommendedCourses] = useState([]);
  const [savedCourses, setSavedCourses] = useState([]);
  const [chatRooms, setChatRooms] = useState([]);
  const [jwt] = useLocalStorageState('', 'jwt');
  const [currentUser] = useLocalStorageState(null, 'currentUser');
  const [documents, setDocuments] = useState([]);

  const navigate = useNavigate();
  useEffect(() => {
    if (jwt && currentUser) {
      httpReqAsync(
        `/api/v1/purchase-records/${currentUser.id}`,
        'GET',
        jwt
      ).then((result) => {
        console.log('result', result);
      });
      httpReqAsync(
        `/api/v1/users/${currentUser.id}/taken-courses`,
        'GET',
        jwt
      ).then((result) => {
        setTakenCourses(result);
      });
      httpReqAsync(
        `/api/v1/users/${currentUser.id}/saved-courses`,
        'GET',
        jwt
      ).then((result) => {
        setSavedCourses(result);
      });
      httpReqAsync(`/api/v1/messages/${currentUser.id}`, 'GET', jwt)
        .then((result) => {
          setChatRooms(result || []);
        })
        .catch((err) => {
          console.log('No chats for the current user', err);
```

Figure 3.4: Data State Management example from Dashboard

### 3.5.2 User Interface and Interaction Design

The UI design focuses on user-friendliness and aesthetic appeal:

- **Responsive Design**: The layout adapts to various screen sizes, ensuring a seamless user experience across different devices. (See figures 3.5, 3.6, 3.7)
- **Interactive Elements**: The UI includes interactive components like search bars, dropdown menus, and real-time chat, enhancing user engagement.
- **Accessibility**: The application maintains accessible design standards, with clear navigation and readable fonts, making it usable for a broad audience.

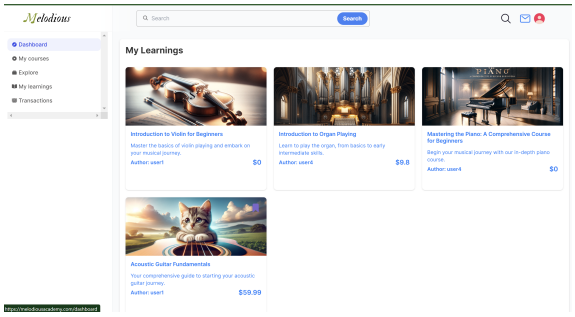


Figure 3.5: Responsitivity: Full layout

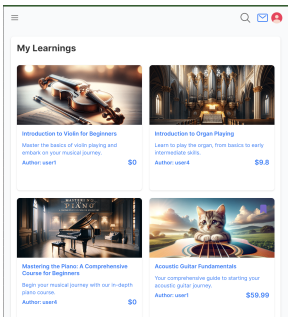


Figure 3.6: Small layout

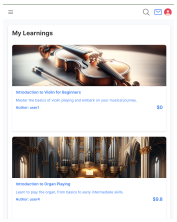

Figure 3.7: Smallest layout

## 3.6 Backend Development

### 3.6.1 Java Spring Boot Setup

The backend of the Melodious application is built using the Java Spring Boot framework, known for its ease of use and efficiency in creating stand-alone, production-grade Spring applications. The setup involves:

- **Project Structure**: Organized into packages such as controllers, services, repositories, and models, ensuring a clean separation of concerns. See figure 3.3 to see the project internal structure.
- **Dependency Management**: Utilizes Maven for managing dependencies, ensuring smooth integration of various libraries and frameworks needed.

- **Application Properties**: Configured in the ‘application.properties‘ file, specifying parameters such as server port, database connection details, and other Spring Boot settings.

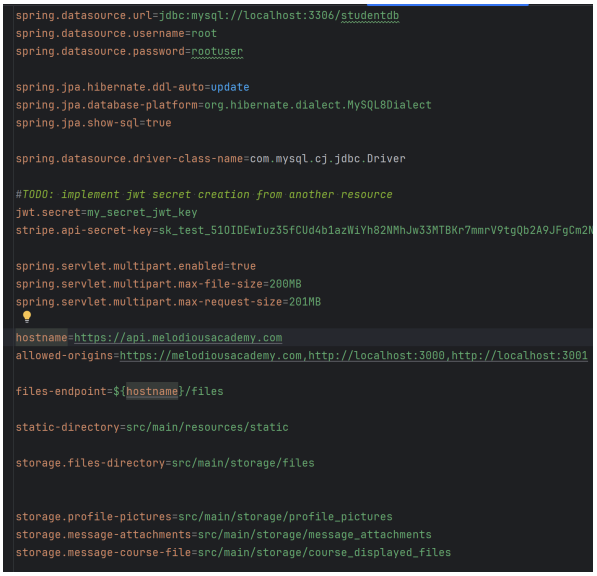


Figure 3.8: App configuration from application.properties file

- **Security Configuration**: Implements Spring Security for authentication and authorization, using JWT for secure token-based user access.

### 3.6.2 Business Logic

The application’s business logic is encapsulated in the service layer, which interacts with the database through repositories and processes user requests received by the controllers. Key aspects include:

- **Controllers**: Handle incoming HTTP requests and respond with appropriate data or status codes. Examples include ‘UserController‘, ‘CourseController‘, and ‘ChatController‘.
- **Services**: Contain core business logic, such as user management, course handling, and chat functionalities. They interact with the database through JPA repositories.
- **Exception Handling**: Systematic handling of exceptions, providing meaningful feedback to users and maintaining application stability.

### 3.6.3 Best Practices and Coding Standards

Adherence to coding best practices and standards is highlighted, with references to Effective Java [5] for Java development guidelines.

### 3.6.4 Database Integration

The backend integrates with a MySQL database, leveraging the Java Persistence API (JPA) for object-relational mapping (ORM). This integration allows for efficient data retrieval and manipulation with minimal boilerplate code.

- **ORM Configuration**: Uses JPA entities to map Java objects to database tables, facilitating data persistence and retrieval operations.
- **Repositories**: Spring Data JPA repositories provide convenient methods for various database operations, abstracting the complexity of direct SQL queries.
- **Data Transactions**: Manages data transactions efficiently to ensure data integrity and consistency across the application.

### 3.6.5 Database Schema Design

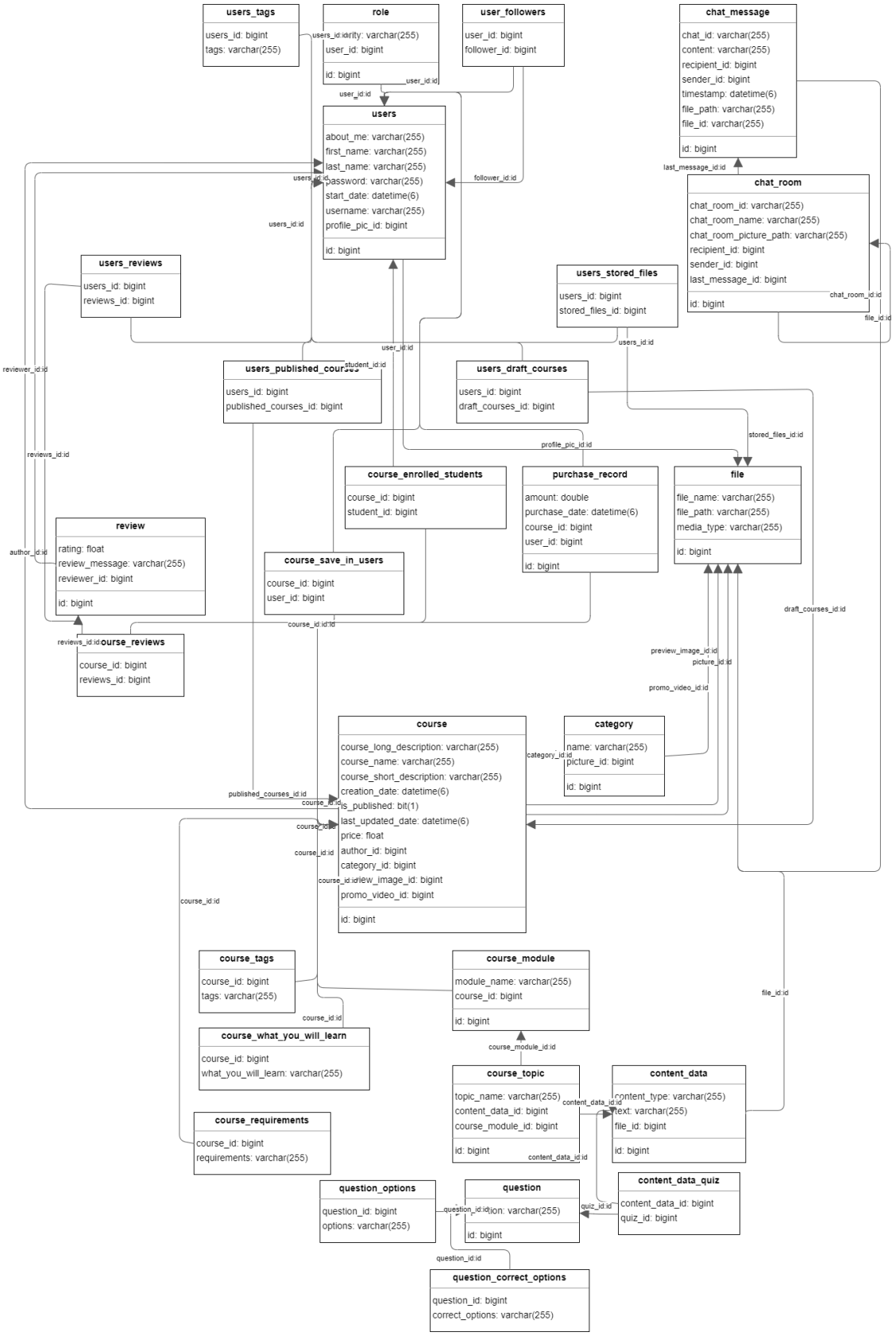


Figure 3.9: Database Schema Diagram

The database schema for Melodious is designed to efficiently manage and relate various entities involved in the platform:

- **User Entity**: Stores detailed user information, including personal details, profile picture, followers, followings, courses published, courses taken, courses saved as drafts, and files stored.
- **Role Entity**: Defines the roles or authorities granted to users, essential for managing access and permissions.
- **Course Entity**: Contains comprehensive details about courses, including author information, enrolled students, users who saved the course, course curriculum, reviews, and associated media files like promo videos and preview images.
- **Category Entity**: Represents course categories, each with a name and associated picture, categorizing courses for easy discovery and organization.
- **Course Module Entity**: Represents individual modules within a course, each containing a collection of course topics.
- **Course Topic Entity**: Contains specific topics within a course module, including the topic name and associated content data.
- **Content Data Entity**: Defines the type of content within a course topic, such as files, documents, images, videos, quizzes, and text.
- **Question Entity**: Represents questions within a quiz, including options and correct answers.
- **File Entity**: Manages file information, including file names, paths, and media types, used across various entities for storing media and document files.
- **Purchase Record Entity**: Tracks financial transactions, associating users with purchased courses, transaction amounts, and purchase dates.
- **Review Entity**: Stores course reviews, including ratings, reviewer information, and review messages.

- **Chat Message Entity**: Manages chat messages, including sender and recipient information, message content, associated file information, and timestamps.
- **Chat Room Entity**: Represents chat rooms, including details about participants, room name, last message, and chat room pictures.

The schema emphasizes normalization to reduce data redundancy and maintain data integrity. Foreign keys and indexing are utilized to optimize query performance and ensure relational integrity among different tables.

This design provides a robust foundation for the platform, ensuring data consistency and supporting complex functionalities like course browsing, user management, and transaction processing.

## 3.7 Security Measures

### 3.7.1 Authentication and Authorization

The Melodious application implements robust authentication and authorization mechanisms to ensure secure access to its features and protect user data:

- **JWT-Based Authentication**: The system uses JSON Web Tokens (JWT) for stateless authentication[6]. Users are authenticated through the ‘/api/v1/auth/login‘ endpoint, where they receive a JWT. This token is then used in the Authorization header for subsequent requests, ensuring secure and efficient user authentication.
- **Role-Based Authorization**: The application implements role-based access control. Users are assigned specific roles, and each role has different access levels and permissions. This approach allows for fine-grained control over what each user can see and do within the application.

### 3.7.2 Data Encryption

To protect sensitive information, the application employs encryption both in transit and at rest:

- **In Transit**: All data transmitted between the client and the server is encrypted using HTTPS and SSL/TLS encryption. This ensures that all data exchanged over the network is secure and protected from eavesdropping or interception.
- **At Rest**: Sensitive data stored in the database, such as user passwords, is encrypted. This protects the data from unauthorized access or breaches, ensuring that even if data is somehow accessed, it remains unreadable and secure.

### 3.7.3 Preventing Security Threats

The application incorporates several strategies to mitigate common security threats:

- **SQL Injection Prevention**: To prevent SQL injection attacks, the application uses prepared statements and Object-Relational Mapping (ORM). This approach avoids the use of raw SQL queries, which can be vulnerable to injection attacks.
- **Cross-Site Scripting (XSS) Protection**: The application sanitizes all user input to prevent XSS attacks. By removing or encoding potentially harmful scripts from user input, it ensures that malicious scripts are not executed in the user's browser.
- **Cross-Site Request Forgery (CSRF) Protection**: CSRF attacks are mitigated by implementing anti-CSRF tokens in forms and requests. This token validation ensures that the requests are genuine and originate from authenticated users.

## 3.8 Testing and Quality Assurance

### 3.8.1 Testing Strategies

- **Unit Testing**: We utilize JUnit5 [7] for unit testing in the backend. This involves testing individual components in isolation to ensure they function correctly. Each module's logic is rigorously tested to maintain code integrity and reliability.

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
new *
public class CourseMapperTest {

    new *
    @Test
    void testMapDtoToCourse() {
        // Setup
        CourseRequestDTO dto = new CourseRequestDTO();
        dto.setId(1L);
        dto.setAuthorId(2L);
        dto.setPrice(99.99f);
        dto.setCourseName("Sample Course");
        dto.setCourseShortDescription("Short Description");
        dto.setCourseLongDescription("Long Description");
        dto.setCategoryId(3L);
        dto.setPromoVideoId(4L);
        dto.setPreviewImageId(5L);
        List<Long> enrolledStudentsIds = Arrays.asList(6L, 7L);
        dto.setEnrolledStudentsIds(enrolledStudentsIds);
        List<Long> savedInStudentsIds = Arrays.asList(8L, 9L);
        dto.setSavedInStudentsIds(savedInStudentsIds);
        dto.setCreationDate(new Date());
        dto.setLastUpdatedDate(new Date());
```

Figure 3.10: JUnit testing setup

```java
// Assertions
assertNotNull(course, message: "Course should not be null");
assertEquals(dto.getId(), course.getId(), message: "Course ID should match");
assertEquals(dto.getAuthorId(), course.getAuthor().getId(), message: "Author
assertEquals(dto.getPrice(), course.getPrice(), message: "Price should match"
assertEquals(dto.getCourseName(), course.getCourseName(), message: "Course nam
assertEquals(dto.getCourseShortDescription(), course.getCourseShortDescriptio
assertEquals(dto.getCourseLongDescription(), course.getCourseLongDescription(
assertEquals(dto.getCategoryId(), course.getCategory().getId(), message: "Cate
assertEquals(dto.getPromoVideoId(), course.getPromoVideo().getId(), message: "
assertEquals(dto.getPreviewImageId(), course.getPreviewImage().getId(), messa
assertTrue(course.getEnrolledStudents().stream().allMatch(user -> enrolledStu
assertTrue(course.getSavedInUsers().stream().allMatch(user -> savedInStudents
assertEquals(dto.getCreationDate(), course.getCreationDate(), message: "Creati
assertEquals(dto.getLastUpdatedDate(), course.getLastUpdatedDate(), message: "
assertEquals(dto.isPublished(), course.isPublished(), message: "Published stat
```

Figure 3.11: JUnit testing assertions

- **API Testing**: Postman[8] is used extensively for API testing. A dedicated Postman workspace is maintained for the project, enabling us to test, document, and share API requests and responses. This helps in verifying the correct behavior of the RESTful services and ensures that the API meets its contract in terms of inputs, outputs, and HTTP status codes.

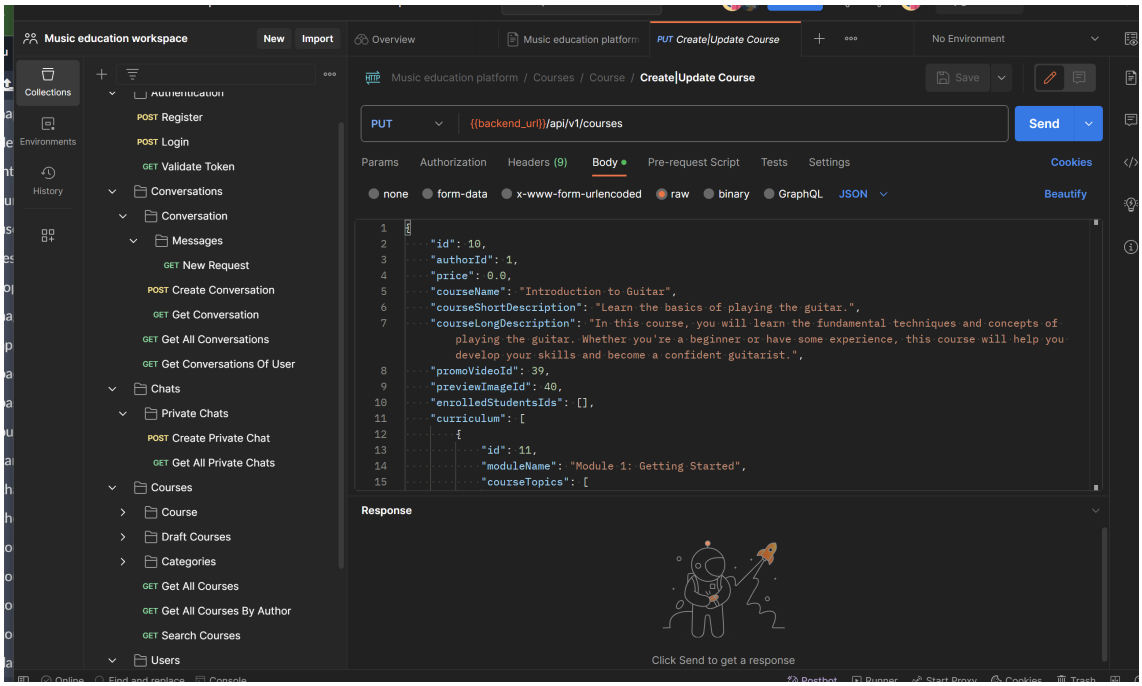


Figure 3.12: Documenting and testing the API using Postman

- **Front-end Testing with Jest**: For our front-end React components, we use the Jest testing framework [**jest**]. Jest provides a robust platform for writing

and running tests on our React components, ensuring they behave as expected. Our Jest setup includes snapshot testing, which helps us track changes in component rendering over time.

src > test > JS ProfilePicture.test.js > ...

```
import React from 'react';
import { render, fireEvent, screen } from '@testing-library/react';
import '@testing-library/jest-dom/extend-expect';
import ProfilePicture from './ProfilePicture';

describe('ProfilePicture Component Tests', () => {

    test('renders with default placeholder image', () => {
        render(<ProfilePicture />);
        const image = screen.getByAltText('Profile');
        expect(image).toBeInTheDocument();
        expect(image.src).toContain('placeholder.png');
    });

    test('renders with provided imageSrc', () => {
        const testImageSrc = 'https://example.com/test-image.jpg';
        render(<ProfilePicture imageSrc={testImageSrc} />);
        const image = screen.getByAltText('Profile');
        expect(image.src).toBe(testImageSrc);
    });

    test('clicking on the image opens modal', () => {
        render(<ProfilePicture />);
        const image = screen.getByAltText('Profile');
        fireEvent.click(image);
        expect(screen.getByAltText('Profile')).toBeInTheDocument();
        expect(screen.getByText('×')).toBeInTheDocument();
    });

    test('clicking close button hides modal', () => {
        render(<ProfilePicture />);
```

Figure 3.13: Testing Profile Picture Component with Jest

- **Manual Testing**: Extensive manual testing has been conducted to cover user functionalities such as account creation, data modification through settings, course enrollment, and Stripe mock payments. This also includes content management like creating, editing, publishing, and drafting courses, as well as ensuring the proper functionality of follow/unfollow features, data CRUD operations, and real-time communication in chat.

- **Future Testing Plans**: While integration and system testing are not yet implemented, they are planned for future development. These testing stages will further ensure the integration of individual modules and the system's overall performance and reliability.

## 3.9 Deployment Process

The process of deployment took several steps and configurations to ensure a reliable and secure hosting environment:

### 3.9.1 Domain and Server Configuration

The website is hosted under the domain 'melodiousacademy.com', which was acquired through GoDaddy. The backend API is accessible via the subdomain 'api.melodiousacademy.com', ensuring a clear separation between the frontend and the backend services.

- **Server Hosting**: The application is hosted on an Ubuntu server provided by VDSina (https://vdsina.ru), chosen for its reliability and the convenience of daily payments. This server is the backbone of our deployment, hosting both the frontend and backend components of the application.
- **DNS Configuration**: Domain Name System (DNS) settings were meticulously configured to correctly point to the hosting server. This includes the configuration of A records and CNAME records to ensure that both the main domain and the subdomain are correctly resolved.
- **SSL/TLS and HTTPS Setup**: To secure data transmission and ensure the privacy and integrity of exchanged data, SSL/TLS encryption was implemented. This setup enables HTTPS for the website, ensuring that all communications between the client and the server are encrypted.

### 3.9.2 NGINX Configuration

NGINX is employed as the web server and reverse proxy for the platform:

- **Web Server Setup**: NGINX serves as the primary web server, handling HTTP requests, serving content, and managing the traffic efficiently.
- **Reverse Proxy Configuration**: As a reverse proxy, NGINX forwards requests to the appropriate backend service, particularly useful for routing requests to the correct application in a microservices architecture.
- **Load Balancing and Performance Optimization**: NGINX also plays a crucial role in load balancing, distributing client requests effectively across multiple server instances. This ensures optimized performance and improved reliability of the application if in the future it will require more servers with more users filling it.

### 3.9.3 Validation

Post-deployment, extensive manual testing was conducted to ensure that all aspects of the server and domain configuration were functional and secure.

# Chapter 4

# Summary

This thesis has presented the development and implementation of Melodious, a comprehensive and innovative platform for music education. Melodious has been designed to provide a user-friendly and engaging learning experience for both students and instructors in the field of music. Through its intuitive interface, diverse course offerings, and interactive features, Melodious stands as a testament to the effective integration of technology in education.

## 4.1 Achievements

Throughout this thesis, significant accomplishments in the development of Melodious have been highlighted. Key features such as a versatile course management system, robust user authentication, and a responsive design contribute to its efficacy as an educational tool. The successful integration of technologies like ReactJS, Java Spring, and Stripe API has enabled a seamless and secure user experience.

## 4.2 Future Improvement Ideas

Looking forward, Melodious has a vast potential for growth and enhancement. Some of the future improvement ideas include:

- **Enhanced Security Features**: Implementation of advanced security protocols to safeguard user data.

- **Artificial Intelligence Integration**: Leveraging AI for personalized course recommendations and adaptive learning paths.
- **Mobile Application Development**: Expanding accessibility through a dedicated mobile app.
- **Gamification Elements**: Incorporating elements like badges, leaderboards, or rewards for course completion could increase user engagement and motivation.
- **Community Features**: Establishing forums or discussion boards for user interaction and support.
- **Live Streaming Classes**: Offering real-time classes for a more interactive learning experience.
- **Offline Access**: Enabling course material access in offline mode.
- **Feedback Mechanism**: Regular collection of user feedback for continuous improvement.

## 4.3 Concluding Words

In conclusion, this thesis project represents not only an academic endeavor but also a personal journey in developing Melodious. I am proud to have achieved most of the objectives I set out with, demonstrating the potential of combining technology and music education in an innovative way. While Melodious currently stands as a personal diploma project, the foundation it lays opens up possibilities for future expansion. With further effort and development, Melodious has the potential to evolve into a significant platform in the field of online music education. For now, it marks a significant milestone in my academic and professional growth, showcasing the skills and knowledge I have acquired and applied in the realm of software development and music education.

# Bibliography


[1] "MySQL 8.0 Reference Manual". In: (2021). URL: https://dev.mysql.com/doc/refman/8.0/en/.

[2] Craig Walls. *Spring in Action*. Manning Publications, 2021.

[3] *Stripe API Reference*. 2021. URL: https://stripe.com/docs/api.

[4] *React - A JavaScript library for building user interfaces*. 2021. URL: https://reactjs.org/.

[5] Joshua Bloch. *Effective Java*. Addison-Wesley Professional, 2018.

[6] *JSON Web Token Introduction*. 2021. URL: https://jwt.io/introduction/.

[7] *JUnit 5 User Guide*. 2021. URL: https://junit.org/junit5/docs/current/user-guide/.

[8] *Postman API Development*. 2021. URL: https://www.postman.com/.

# List of Figures

# List of Tables