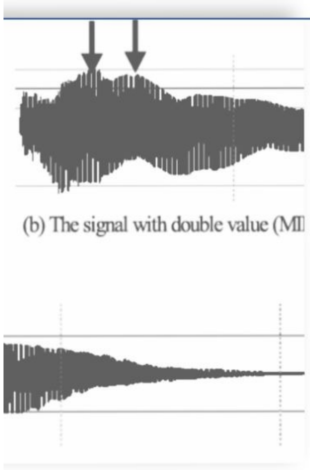
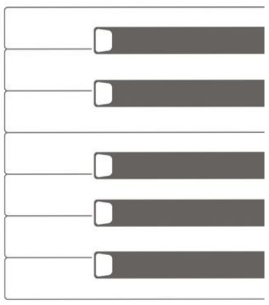
# Low-Level Design Report



# pianissimo

Team Members:
Melisa Kamacı
Tamer Alkım Tokuç
Nazem Yıldırım
Begüm Hatipoğlu

Supervisor: Yücel Çimtay
Jury Members:
Emin Kuğu & Venera Adanova



## 1. Introduction

Pianissimo is a desktop application that fills a critical need for pianists and piano enthusiasts who require an efficient and reliable solution for transcribing solo piano music into sheet music. The ability to transcribe piano music from audio recordings is a valuable skill, but it can be time-consuming and error-prone. Pianissimo provides a trusted and practical solution to this problem, allowing musicians to transcribe music quickly and accurately.

The need for Pianissimo arises from the growing demand for music transcription services in the music industry. As previously mentioned, transcribing music from piano compositions is an essential skill for music educators, students, and performers. Since this need had to be automated, Pianissimo took the stage and came up with an accessible and cost-effective way to transcribe piano music, making it an ideal choice for musicians of all skill levels.

This report will provide an in-depth analysis of the system's design and functionality, including the various modules, components, and data structures that make up the application. In other words, this report will provide a comprehensive overview of the low-level design of Pianissimo, including its internal architecture, class interfaces, packages and content that root from related topics. By analyzing the application's design at this level of detail, we aim to provide a detailed understanding of how the system functions and how it can be further improved to meet the needs of its users.

### 1.1. Object Design Trade-offs

This section will discuss the object design tradeoffs that were considered through the planning of implementation to make sure that the system would satisfy its functional and performance objectives while still having a manageable codebase.

#### 1.1.1. Execution Time and Ease of Use

One significant trade-off was the selection of the programming language and packages. Python was chosen as the primary programming language for Pianissimo because of its powerful libraries for audio processing such as librosa, and its ability to create desktop applications with a user-friendly interface using Tkinter. However, the use of Python also introduces some performance trade-offs compared to languages like C++, which can be faster for certain operations.

### 1.1.2. Accuracy and Speed

Another trade-off considered during the object design phase was the trade-off between accuracy and speed. Pianissimo's core functionality is to provide accurate transcriptions of piano music from audio files. To achieve this, the application uses complex algorithms and data structures to analyze the audio recordings containing the piano music to identify notes, and other musical elements. However, this level of accuracy comes at a cost in terms of processing time. Therefore, Pianissimo's design strikes a balance between accuracy and speed by using optimized short time fourier transform resolution and window size while still protecting the level of accuracy. Increasing resolution and lowering window size does not provide significant accuracy improvement, and is selected as current values to keep Pianissimo accurate and while still providing decent speed.

### 1.1.3. Flexibility and Simplicity

The object design phase also considered the trade-off between flexibility and simplicity. Since SOLID principles are in focus of the project, each class has a single responsibility. Separate interfaces for GUI, database connection, file conversion, note detection, and output formatting will be created. These classes are reusable in other similar projects as needs are common. This comes with increased simplicity cost, as more classes make the project complex. In addition, interfaces will be added to keep classes segregated, and this will increase complexity as well. Overall, flexibility and reusability is preferred over simplicity

### 1.1.4. Performance and Safety

Performance and safety are both evaluated when deciding on design steps of Pianissimo. Since high value personal data is not held in the system, performance is given more importance. Still, sufficient measures to keep passwords secure have been taken, such as encryption. This will ensure that user data is safe while the system is fast enough to keep up with user requests. The music data in the databases are not encrypted as they do not contain personal data, and to ensure performance of the system.

### 1.1.5. Extensibility and Performance

By the nature of the application, Pianissimo is an application that is open to potential feature extensions such as advanced music analysis, integration with other music notation software, customization, sharing and editing options. However, each of these extensions bring more complexions and potential performance issues. For instance, advanced music analysis features would require more computational power and processing time and would increase the response time of the application. Integration with other music notation software and customization-sharing options would require further network communication and system resources which could cause overhead and decrease in the performance. Editing of the input audio files would also require further processing power and memory. It is crucial to implement and optimize potential feature extensions carefully and use the most efficient algorithms and data structures in order to balance performance and functionality.

### 1.1.6. Compatibility and Extensibility

Potential extensions for the application may also result in compatibility issues such as issues with operating systems, hardware configurations and audio file formats. If we are to exemplify from the potential feature extensions mentioned in section 1.1.5; these extensions could introduce a myriad of compatibility issues if the system is not properly tested and optimized considering different audio file formats, operating systems, hardware configurations, software packages, cloud storage providers and internet speeds or configurations. Similarly, with careful implementation and apprehensive testing, the compatibility issues should be minimal.

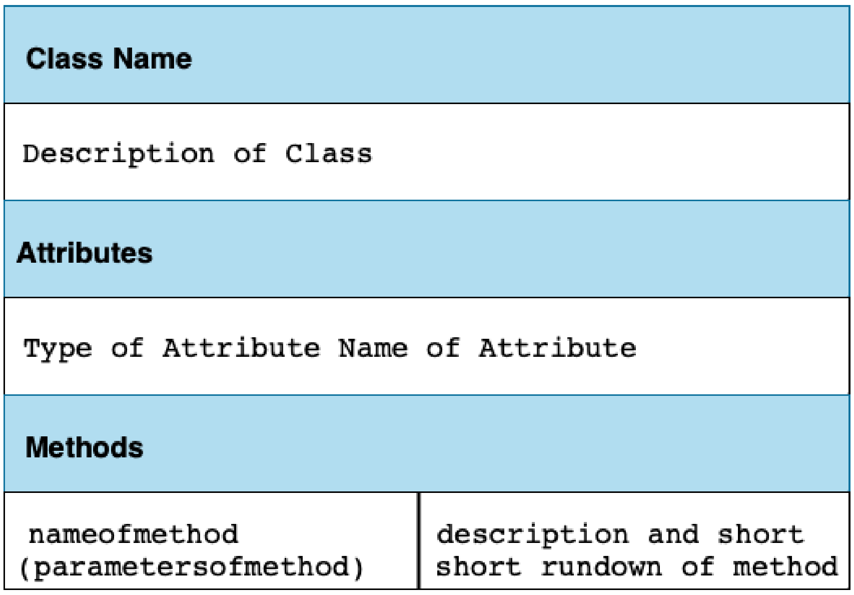
### 1.1.7 Quality Assurance and Time

The quality assurance process requires more time if a project has multiple criteria. More time was spent to meet these criteria during the implementation of Pianissimo. The process is managed in a quality manner, there is no loss of time.

Overall, the object design phase of Pianissimo involved several trade-offs between performance, accuracy, flexibility, simplicity, performance and safety. These trade-offs were carefully considered to ensure that the application meets its functional and performance requirements while maintaining a maintainable codebase and user-friendly interface.

## 1.2. Interface Documentation Guidelines

The report has been organized as follows. There are given class names. The names of variables and methods are variable name and method name.(). In the below figure, the class name is listed first, then the class's attributes, and finally the class's methods.



| Class Name | |
|---|---|
| Description of Class | |
| **Attributes** | |
| Type of Attribute Name of Attribute | |
| **Methods** | |
| nameofmethod (parametersofmethod) | description and short short rundown of method |

Separate tables for all classes and attributes are detailed below.

| **Class** | App |
|---|---|
| Main GUI management class | |
| **Attributes** | |
| passwordInput | String. Entered password for encryption |
| **Methods** | |
| __init__(self, *args, **kwargs) | Initializes the application |
| remove_placeholder(self, input, title) | Remove placeholder text after input is entered, void. |
| put_placeholder(self, input, title) | Place placeholder back if no input entered, void. |

| | |
|---|---|
| remove_password_placeholder(self, passwordInput) | Remove placeholder text after password is entered, void. |
| put_password_placeholder(self, passwordInput) | Place placeholder back if no password entered, void. |
| encrypt_password(self, password) | Encrypts and salts password before sending it to the database, string |

| **Class** | AuthorizationPage (Child of App class) |
|---|---|
| Login page of the app | |
| **Attributes** | |
| CANVAS_WIDTH | Int. Width of window with 900 px constant value. |
| CANVAS_HEIGHT | Int. Height of window with 550 px constant value. |
| clickCount | Int. Amount of clicks on password hide button. |
| emailInput | String. Holds the value of email entered by the user. |
| passwordInput | String. Holds the value of the password entered by the user. |
| **Methods** | |
| show_password() | Sets password visible, void |
| login() | Attempts login, void |
| register() | Opens register page, void |

| **Class** | RegisterPage (Child of App class) |
|---|---|
| Login page of the app | |
| **Attributes** | |
| nameInput | String. Holds the name value entered by the user. |
| surnameInput | String. Holds the surname value entered by the user. |
| emailInput | String. Holds the email value entered by the user. |
| passwordInput | String. Holds the password value entered by the user. |
| **Methods** | |
| go_back() | Returns to login page, void |
| register(name, surname, email, password) | Registers user to the database, void |
| show_password() | Makes password visible, void |

| **Class** | PreviousConversionsPage (Child of App class) |
|---|---|
| Window to display previous conversions | |
| **Attributes** | |
| conversions | List. Previous conversions of users |
| **Methods** | |
| display() | Opens selected conversion window, void. |
| download() | Opens selection menu to choose MIDI or PDF and initiates download, void |

| **Class** | NewConversionPage (Child of App class) |
|---|---|
| Input page of the app. | |
| **Attributes** | |
| fileLocation | String. Directory of the file to be converted. |
| songName | String. Name of the piece. Empty if useFileName is true |
| songComposer | String. Name of composer of the piece. Empty if useFileName is true |
| audioFile | File. Audio file |
| useFileName | Boolean. True if “Use file name” box selected |
| validity | Boolean. True if provided file is supported |
| **Methods** | |
| is_valid(audioFile) | Returns true if format is supported, boolean. |
| name_page(validity) | If the provided file is valid, go to the naming page. |
| start_conversion(audioFile) | Sends the file to NoteDetection class, void. |
| upload_conversion(conversion) | Posts conversion to the database, int. |

| **Class** | ConversionProgressPage (Child of App class) |
|---|---|
| Display progress of current conversions | |
| **Attributes** | |
| conversionID | Int. ID of conversion. |
| conversions | List. conversions done by the user. |
| percent | Int. Completed percent of conversion. |
| **Methods** | |
| display_coversions() | Displays conversions returned from the database. List |

| **Class** | db |
|---|---|
| Database connection manager | |
| **Attributes** | |
| databaseURL | String. Database connection string retrieved from .env file |
| client | String. Connected client ID. |
| db | String. Database model. |
| **Methods** | |
| create_user(name, surname, email, password) | Creates users according to provided information. Int |
| retrieve_coversions() | Retrieves users' conversions from the database. List |
| upload_conversions(conversion) | Uploads conversion to the database provided by |

| | |
|---|---|
| | NewConversionsApp class. Int |

| **Class** | NoteDetection |
|---|---|
| Class to create note data. | |
| **Attributes** | |
| sampleRate | Int. Encoding sample rate of the file. Retrieved from the file. |
| audioData | Numpy.ndarray. Signal values of the file. |
| notes | List. Notes detected from detectNotes method. |
| **Methods** | |
| detect_note(frequency) | Matches frequency of audio file with frequency chart retrieved from note_frequencies.csv file to determine notes. |
| detect_notes(audio_data, sample_rate) | Applies fourier transform to audio data to take frequencies. Uses sample rate to apply fourier transform to each window. |
| apply_filtering(audio_data, sample_rate) | Applies band pass filter to eliminate frequencies below 16 Hz, and above 7900 Hz since the range of piano is between these values. |

| **Class** | FileConversion |
|---|---|
| If the input file is not in wav format, this class transforms it to wav to be able to detect notes. | |
| **Attributes** | |
| inputFormat | String. Current format of input. |
| **Methods** | |
| mp3_to_wav(file) | Converts MP3 format to WAV format |
| aac_to_wav(file) | Converts AAC format to WAV format |
| flac_to_wav(file) | Converts FLAC format to WAV format |

| **Class** | MIDIOutputFormatter |
|---|---|
| Creates MIDI output | |
| **Attributes** | |
| piece | Object. Notes inferred from the input. |
| converted | List. Notes converted to fit MIDI format. |
| **Methods** | |
| find_pitch(note) | Converts note to MIDI format pitch |
| find_time(note) | Finds starting time of note and converts it to MIDI format time |
| find_duraiton(note) | Converts note length to MIDI format duration |
| compose(converted) | Composes and creates MIDI output from converted notes. |

| **Class** | PDFOutputFormatter |
|---|---|
| Creates PDF output | |
| **Attributes** | |
| piece | Object. Notes inferred from the input. |
| **Methods** | |
| compose(piece) | Creates LaTeX code to output PDF file. |

### 1.3. Engineering standards (e.g. , UML and IEEE)

In Pianissimo, PEP8 styling standard is used. Some of the important rules are; indentation is done with 4 spaces, classes are in pascal case, variables are camel case, constants are screaming snake case, functions are snake case. These conventions are strictly applied to ensure readability.

As mentioned in the report, the Unified Modeling Language (UML) standard was used to model class interfaces and interactions. UML is a very important part of object-oriented software development and the software development process. The first of these standards is through object, class, package and deployment diagrams, and the second is through activity and sequence diagrams. Mainly, UML instructions are used for instances, subsystem compositions, and hardware descriptions used to describe class interfaces, diagrams, scenarios.

In Pianissimo's reports, the Institute of Electrical and Electronics Engineers (IEEE) style was followed in the reference and reference phase.

### 1.4. Definitions, Acronyms, and Abbreviations

- Sheet music: A transcription or printing of a musical composition that includes the lyrics, harmony, and melody (if applicable). Standard notation, which uses symbols and notation to express various pitches, rhythms, and other musical elements, is generally used to write sheet music.

- Solo Piano: A musical composition or performance that only features a piano player. Without any additional instruments or voices, solo piano music mainly consists of melodies and harmonies played on the piano.
- Transcription: The process of converting a piece of music from one form (such as audio) into another form (such as sheet music). Transcription is typically done manually by a musician or using specialized software.
- MIDI (Musical Instrument Digital Interface): A standardized protocol for electronic musical instruments, computers, and other devices to communicate and exchange information. MIDI files are digital representations of music that can be played back using a MIDI-compatible device.
- PDF (Portable Document Format): A file format for representing documents in a platform-independent manner. PDF files can be viewed and printed on a wide range of devices and software applications.
- Signal Processing: The field of electrical engineering that involves the analysis and manipulation of signals, such as audio signals or images. Signal processing techniques are often used in the analysis and processing of audio files.
- Audio File: A digital representation of an audio recording, such as a song or piece of music. Audio files can be in various formats, including MP3, WAV, and AIFF, and can be played back using specialized software or hardware.
- Note: In music notation, a symbol representing a specific pitch and duration that is played or sung as part of a musical composition. Notes are typically represented on sheet music using standard notation, with different symbols and notation used to represent different pitches, durations, and other musical elements.

## 1.5 Overview

Pianissimo was designed as a computer program that converts solo piano sound recordings into sheet music. The main purpose of Pianissimo is to make it easy for users to access accurate and professional quality sheet music for their favorite piano pieces without the need for extensive sheet music knowledge or the use of special software.

To create notes, Pianissimo uses techniques to analyze sound and determine musical characteristics such as pitches, rhythms and sentences.

In the program, this information is converted into a format that can be viewed as sheet music. To ensure the accuracy of the created notes, Pianissimo also provides mechanisms for user input and correction.

Once the notes are created, they can be viewed within the program and downloaded in various formats such as PDF or MIDI. Pianissimo also includes tools for users to customize and edit notes, including adjusting their layout and format. This allows musicians to tailor the notes to their specific needs and preferences.
Pianissimo users can create their own accounts and customize their preferences within the app. All this account information will be stored securely using a database system. This allows users to access their notes and preferences from any device and makes it easy to keep track of multiple pages and projects. Emphasis is placed on performance and safety.
Overall, Pianissimo is designed to be a comprehensive and user-friendly resource for solo piano sheet music and offers an alternative to traditional music notation software.

## 1.6 Libraries

- Tkinter: The tkinter package is the standard Python interface to the Tcl/Tk GUI toolkit. This package provides GUI building tools which are necessary to interact with users. Additional toolboxes, such as tkinter.font are used for extended usage and design.
- Bcrypt: Password hashing package sufficient for mild applications. For our system, to ensure system speed, only passwords are encrypted. Additional salting is used for increased security.
- Db: db is an easier way to interact with databases. It is used to explore tables, columns, views.
- Pymongo: Pymongo is a Python package that provides a convenient interface for interacting with MongoDB. Pianissimo uses pymongo to connect to

MongoDB, query and manipulate data stored in MongoDB, and perform various administrative tasks.

- Dotenv: Dotenv reads key-value pairs from a .env file and can set them as environment variables.
- OS: This module provides a portable way of using operating system dependent functionality. This package is used for the .env file which is necessary for communicating with the database.
- Re: Re reads regular expressions. This package helps to confirm email data entered to GUI is, in fact, an email.
- Numpy: NumPy is the fundamental package for scientific computing in Python. In the project, functions related to fourier transform, detection of strongest signal in a given time window, and reducing array found by taking input to one-dimensional array.
- Matplotlib.pyplot: Pyplot toolbox of Matplotlib library is used to display resulting spectrograms of short time fourier transform. This library will not be used in the end product and will be used for development purposes only.
- Pandas: Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. In the project, read_csv method to read note_frequencies csv file.
- Librosa: Librosa is a Python package for music and audio analysis. It provides tools for a wide range of tasks in audio processing, including loading and saving audio files in various formats, preprocessing audio data, extracting features from audio data, computing beat and tempo information and estimating pitch and tuning information.
- MIDIUtil: MIDIUtil is a pure Python library that allows one to write multi-track Musical Instrument Digital Interface (MIDI) files from within Python programs. This package will help to output a MIDI file if selected by the user.
- Abjad: Abjad is a Python API for Formalized Score Control. Abjad is designed to help composers build up complex pieces of music notation in an iterative and incremental way. To output a PDF file, this package will be used for providing an output.

- Scipy: SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. 2 toolboxes from Scipy package are used;
    - Scipy.signal: Scipy toolbox related to signal processing. This package is the next fundamental building block of Pianissimo as the project is based on signal processing. Functions related to short time fourier transform are used in the project.
    - Scipy.io: Scipy toolbox related to file input. Wavfile read function is used to handle primary input extension wav.

## 2. Packages

Since the Pianissimo application uses the MVC design pattern, the client side of the application mainly contains the View component which consists of the interfaces the users interact with. The controllers provide the communication between the client and server; therefore it can be said to be in both client and server sides.

### 2.1 View

The view part consists of 5 components:

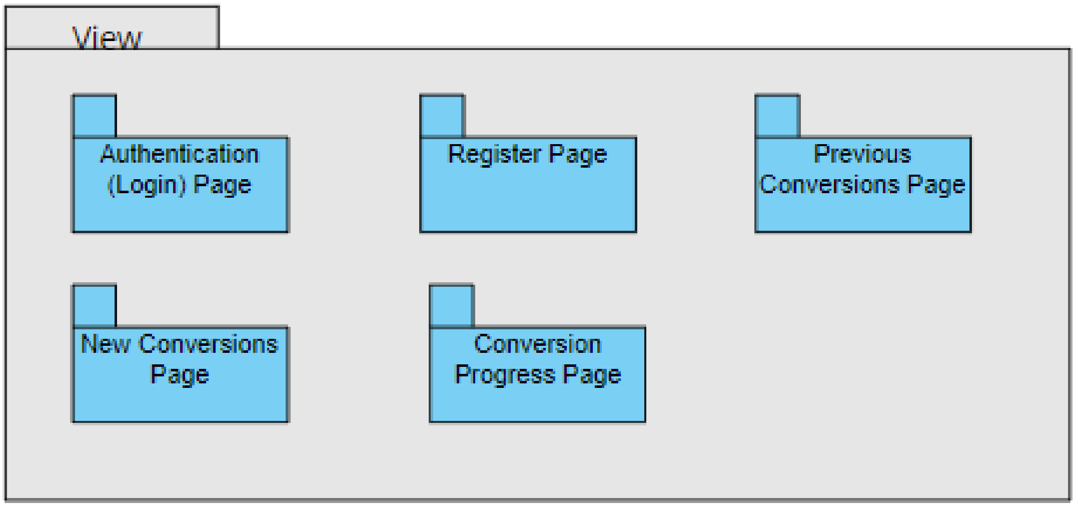
Authentication page, which prompts the user to enter their login credentials which then verifies the user's identity by checking the credentials against the database.

Register page, which allows users to create a new account by providing a valid email address and a password. The email address and the encrypted password then gets stored in the database.

Previous conversions page, in which the user is able to view their previously converted audio file to music sheets.

New conversions page, in which the user is able to upload a new audio file in order to convert it to a music sheet.

Conversion progress page, in which the user is able view the current progress of the files that are in the process of conversion.

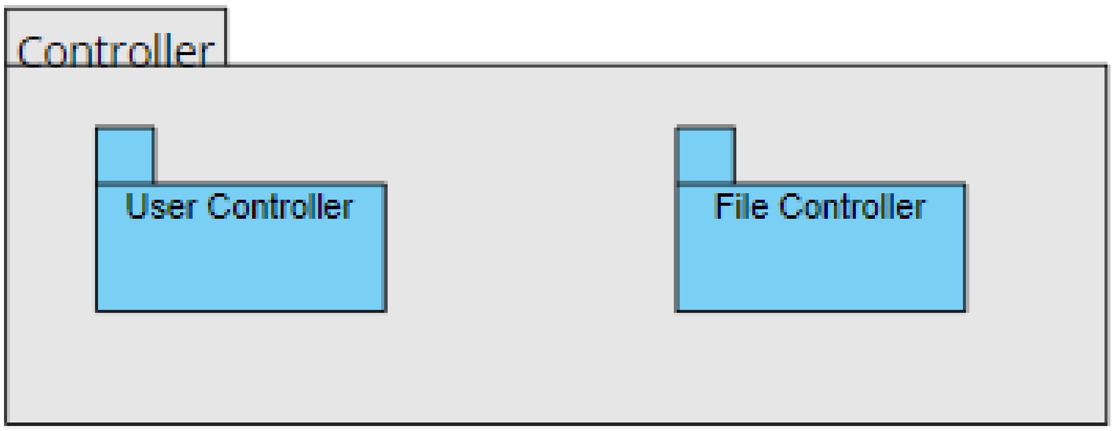


### 2.2 Controller

Since the note detection and the conversion from the audio file to sheet music are done on the client side, the controllers are mainly used for user account management and conversion files' management.

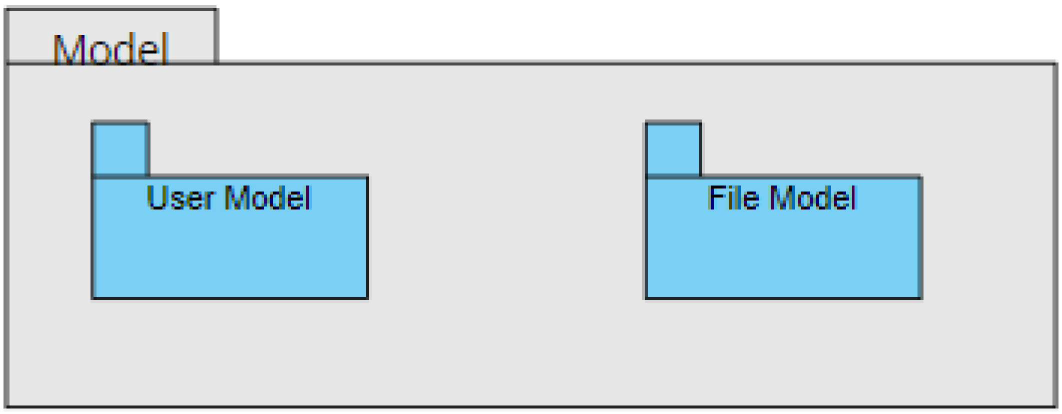
User Controller, the user controller will have methods for managing user profiles and dealing with authentication. Using their credentials, the user can retrieve user information (like their role and permissions) and update user information (such as their profile information or password).

File Controller, the conversion file controller is responsible for handling incoming requests for file conversions, overseeing file data and the conversion procedure, and providing the user with the converted file.



### 2.3 Model

The main models are User and File models which will contain the information related to the entities.



## 3. Proposed Software Architecture

### 3.1 Overview

The software architecture part, which combines the structures and contains specific details about the system structure of the application Pianissimo, is explained in the analysis report (prior reports). The relationships between the subsystems and the future technical ambitions are detailed in the subheadings.

## 3.2 Subsystem Decomposition

Pianissimo’'s large-scale architecture adopts the MVC (Model-View-Controller Architecture) model. It will be in the form of a computer application interacting with a single user and will allow the user to use notes in PDF or MIDI format whose voice is converted into notes using signal/audio processing techniques. It includes most of the data processing required to convert uploaded audio files to PDF, such as signal detection of notes.

Our system is built on a Model-View-Controller architectural model. We have divided our system into three subsystems; PianissimoModel (Model), PianissimoView (View), PianissimoController (Controller). The PianissimoModel subsystem is responsible for maintaining the domain information and the content of the entire Pianissimo view. The PianissimoController subsystem is responsible for running controls, creating the application layout, controlling the rules, and executing the processing throughout the execution of pianissimo and sound transform to (PDF/MIDI) . It depends on the PianissimoModel. The PianissimoView subsystem is responsible for the execution interface and for managing the order of interactions with the user. It is up to the Pianissimo Controller to provide information to continue updating the app's view.

## 3.3 Hardware Software Mapping

The physical deployment of software artifacts on deployment targets is accomplished using this methodology. In other words, it shows how the components are distributed during operation.

The communication between the user's device and the input/output devices for the Pianissimo computer program can be used to map the project's hardware and software. The user device's input controller is in charge of managing each type of input that the computer application accepts and transmits to the keyboard or mouse on the IO Device, as can be seen in the figure. The app's UI makes it easy for the user to mark.

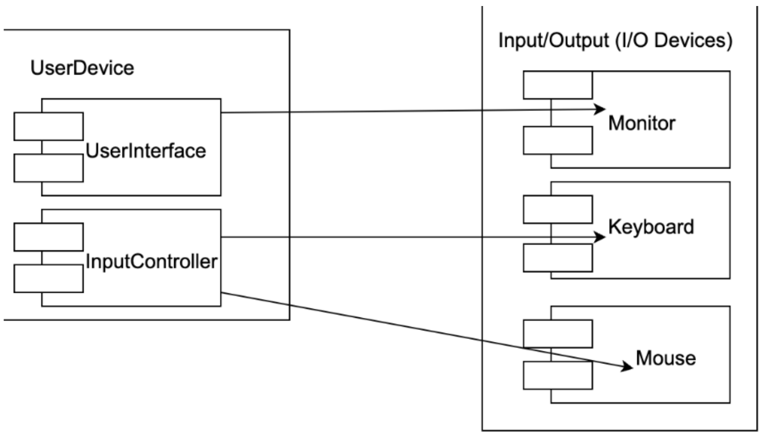
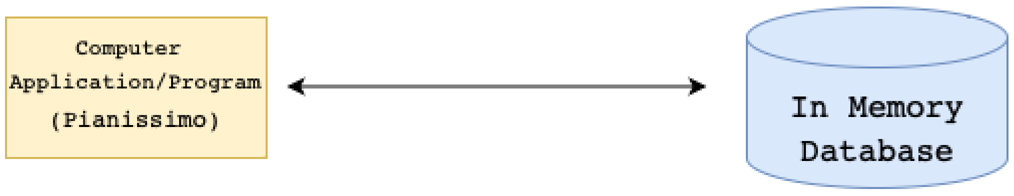


Diagram showing the components used in Hardware/Software Mapping communications between subsystems.

### 3.4 Persistent Data Management



Persistent data includes both user-taken notes (in PDF or MIDI format) and information held in the database. When a user saves an audio file, the note is saved locally in the database remotely if the recorded audio file was uploaded via the page. Audio files are saved in PDF or MIDI format. Notes can remain in the database as long as their total size does not exceed the limit provided by the PianissimoModel (database). As stated, a maximum of 50 projects will be stored in the system and a maximum of 100 users will be able to log in at the same time. Also, information about users and notes will be kept in MongoDB.

This information is also used for services such as authentication. Out-of-the-box trained models are also kept on the database side and used whenever a module that uses them is called. Notes will be built so that it can use local storage to store data about user data like Ubuntu 18.10. A database system is used to store notes, authentication data, and other data about users. MongoDB, a very reliable and flexible database engine, was chosen as the database management system.

## 3.5 Access Control and Security

Users must create an account to use Pianissimo. For authentication services, each user must choose a username and password and email address. The user can register with Pianissimo to create a profile. The user can log in to view their profile and previously converted notes, if any. A user can log out directly after logging into the pianissimo application page. Users can upload notes before logging out.

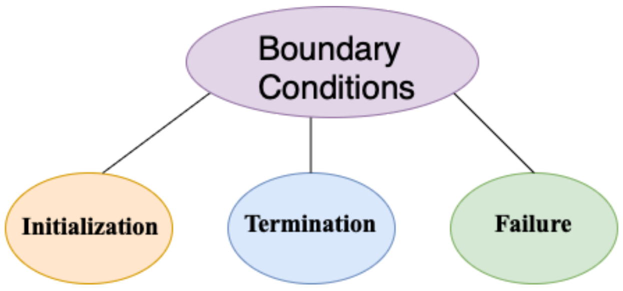
The user can download the output of the notation file or print it without downloading it. The user can retrieve and download an existing music file from the user's database. The user can remove a selected note file from their profile that they have converted before. Users will be asked to create strong passwords when creating their account. In addition, users will have accepted the terms and conditions to be established in accordance with local regulations such as general data protection regulation KVKK. Additionally, data security is a crucial concern that needs to be rigorously maintained. We created our program to comply with the General Data Privacy Regulation (GDPR).

## 3.6 Global Software Control

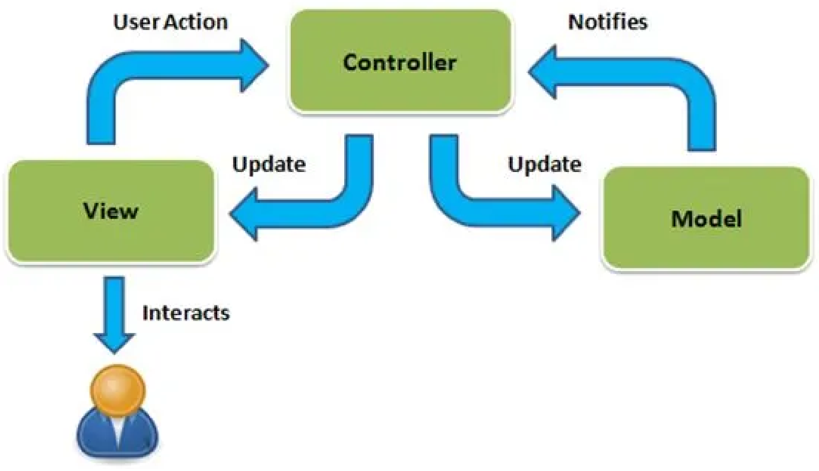
Pianissimo has a software control with MVC Architecture. Applications are managed through event driven software control in response to user input. User selections in Pianissimo pass requests to the application while creating a transcript (PDF/MIDI) from the audio file. When the application processes the requests and returns output, the user is given the rendered result. Some of the data provided is not used by users.

For example, if a user enters a valid email address and password, the application will receive an authentication token that will be used by the program to open the main menu screen. As a result, this event-based checking is also performed in other areas, including authentication, where a login request is made using the username and password that the user enters, and an authentication token is returned if the credentials are correct. After successful authentication, a request is made asking the user to list their notes. Another example would be a response to an information update request whenever a user changes a preference.

## 3.7 Boundary Conditions



Three boundary conditions, including initialization, termination, and failure, will be present in Pianissimo. Developers evaluate each component and investigate failures using boundary conditions. These circumstances enable programmers to handle program faults. Boundary conditions, in other words, Pianissimo how the user and developer should behave in relation to the system's original behavior.



### 3.7.1 Initialization

Since Pianissimo will be a computer application based application, users must have a device to download and use the application. That is, users need an internet connection and a computer to use the Pianissimo computer application. The application can be accessed via the application interface.

### 3.7.2 Termination

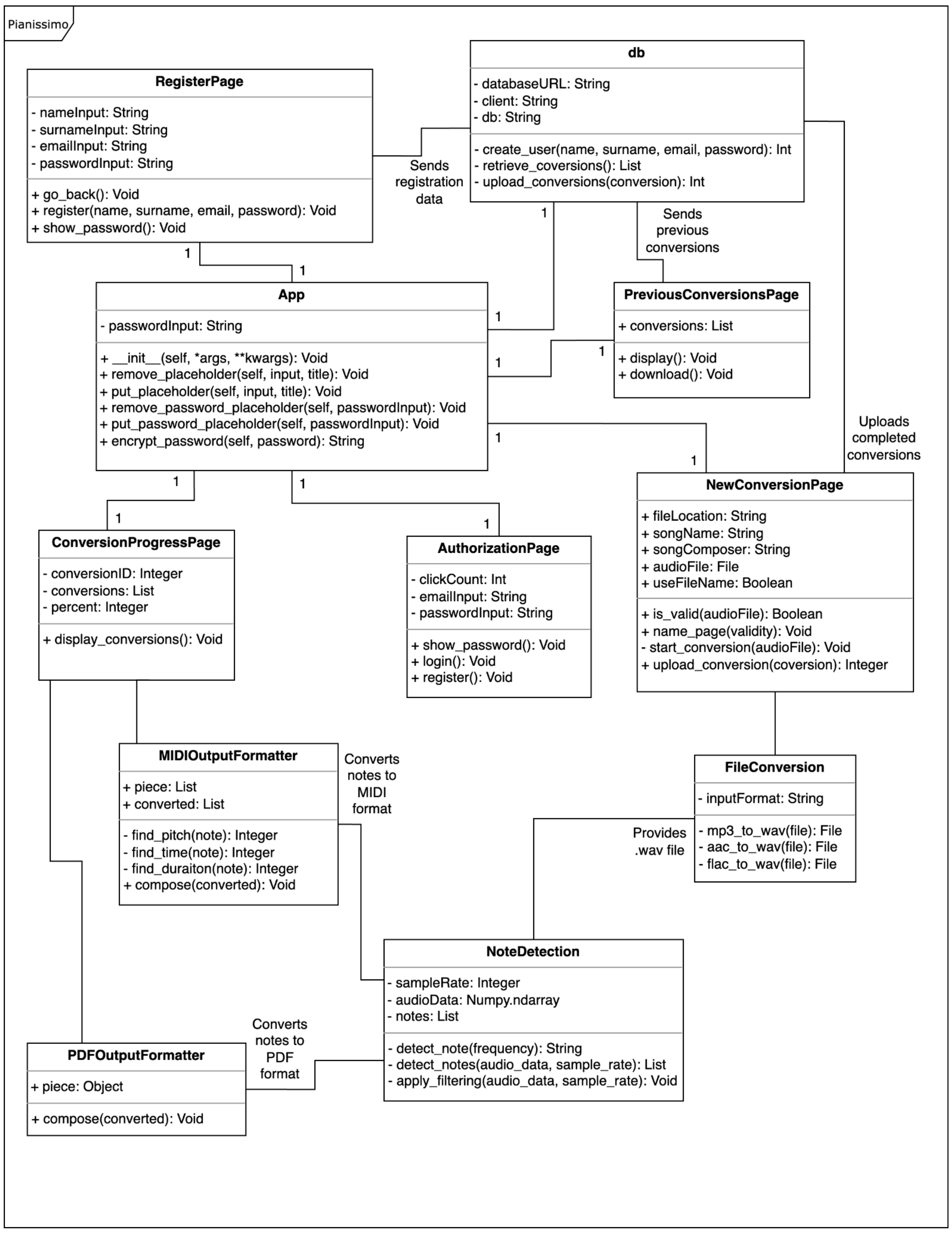
Logging out (log out) in Pianissimo is simple since users can exit the application at any time by closing the application with the buttons provided by the application page or the exit button on the home screen (from the settings tab at the bottom left). The user can log out of Pianissimo at any time, except when waiting for the result of an operation on a note file in the application.

### 3.7.3 Failure

Since some features of Pianissimo require an internet connection, no internet connection can cause malfunctions (sometimes). Also, in some error situations, the user may need to close and reopen the application. In the event of a problem with Pianissimo, an attempt is made to save the relevant system data before it is terminated. Since the application has been closed, users must log in again. Since all completed transactions are recorded in the database, there will be no data loss if the application fails while the user is performing the operation. This incomplete action will not be saved, so the action needs to be done again after the user reopens the app. A stable internet connection is important when using the application, as it provides the connection between the Internet and the computer application. Pianissimo may fail if the user device loses the internet. Additionally, Major Errors in Pianissimo will be capped at a maximum of 0.25 errors/KLOC, including system failures, security breaches, and failure to produce output. It is stated that critical errors such as users with excessive or insufficient access will be limited to a maximum of 1 error/KLOC.

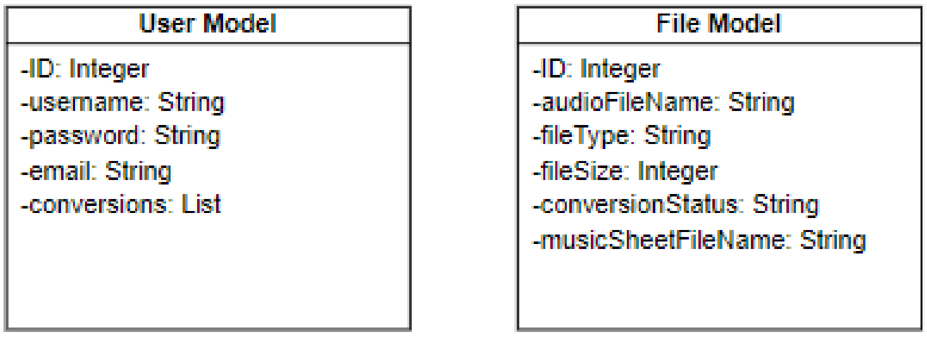
## 4. Class Interfaces

Class interfaces can be displayed using UML class diagrams. This illustration suggests leaving scaffolding code out of class interfaces. For private attributes, getter and setter methods are therefore not explicitly displayed. For their private attributes, all classes have public getter and setter methods. General display of all packages is in the figure.


Pianissimo
RegisterPage
- nameInput: String
- surnameInput: String
- emailInput: String
- passwordInput: String
+ go_back(): Void
+ register(name, surname, email, password): Void
+ show_password(): Void
db
- databaseURL: String
- client: String
- db: String
- create_user(name, surname, email, password): Int
- retrieve_coversions(): List
- upload_conversions(conversion): Int
Sends registration data
Sends previous conversions
Uploads completed conversions
App
- passwordInput: String
+ __init__(self, *args, **kwargs): Void
+ remove_placeholder(self, input, title): Void
+ put_placeholder(self, input, title): Void
+ remove_password_placeholder(self, passwordInput): Void
+ put_password_placeholder(self, passwordInput): Void
+ encrypt_password(self, password): String
PreviousConversionsPage
+ conversions: List
+ display(): Void
+ download(): Void
NewConversionPage
+ fileLocation: String
+ songName: String
+ songComposer: String
+ audioFile: File
+ useFileName: Boolean
+ is_valid(audioFile): Boolean
+ name_page(validity): Void
- start_conversion(audioFile): Void
+ upload_conversion(coversion): Integer
ConversionProgressPage
- conversionID: Integer
- conversions: List
- percent: Integer
+ display_conversions(): Void
AuthorizationPage
- clickCount: Int
- emailInput: String
- passwordInput: String
+ show_password(): Void
+ login(): Void
+ register(): Void
MIDIOutputFormatter
+ piece: List
+ converted: List
- find_pitch(note): Integer
- find_time(note): Integer
- find_duraiton(note): Integer
+ compose(converted): Void
Converts notes to MIDI format
FileConversion
- inputFormat: String
- mp3_to_wav(file): File
- aac_to_wav(file): File
- flac_to_wav(file): File
Provides .wav file
NoteDetection
- sampleRate: Integer
- audioData: Numpy.ndarray
- notes: List
- detect_note(frequency): String
- detect_notes(audio_data, sample_rate): List
- apply_filtering(audio_data, sample_rate): Void
PDFOutputFormatter
+ piece: Object
+ compose(converted): Void
Converts notes to PDF format
1
1
1
1
1
1
1
1
1
1
1
1
1
1

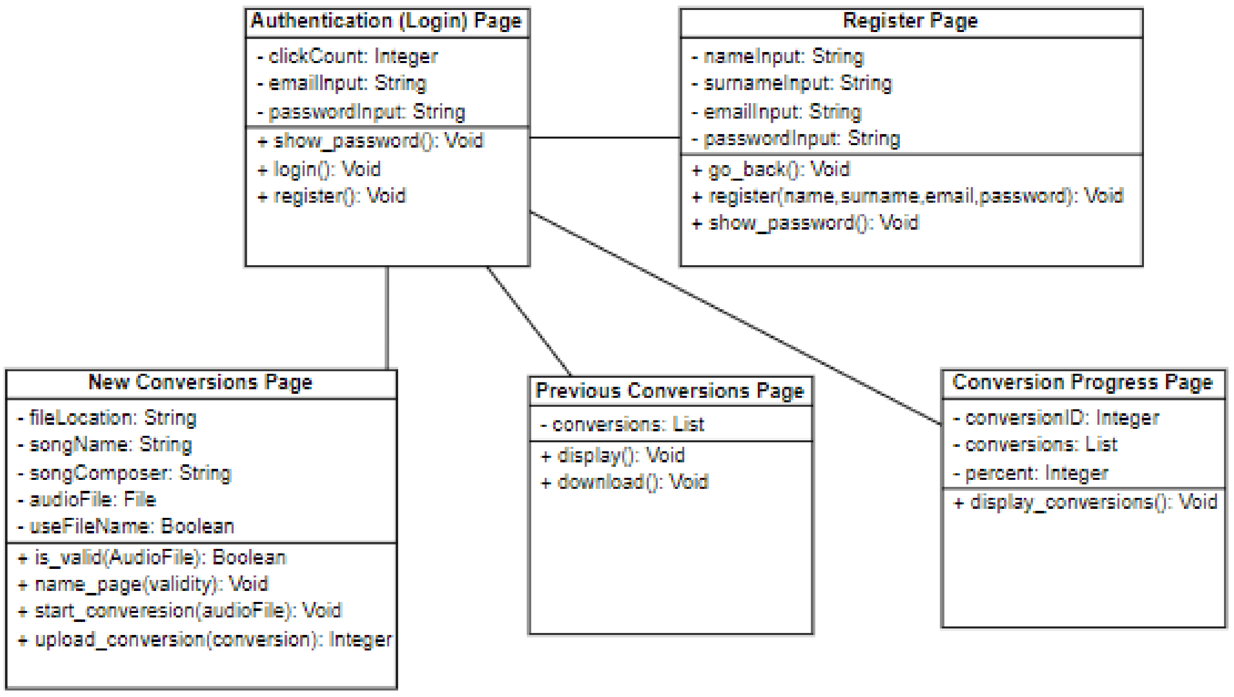
## 4.1 Model Package

The components in the model package are responsible for managing user and file data such as usernames, passwords, email addresses, file names, audio file names, file size and types. The models also validate user registration and file inputs. The model package ensures user and file data are stored securely and efficiently.



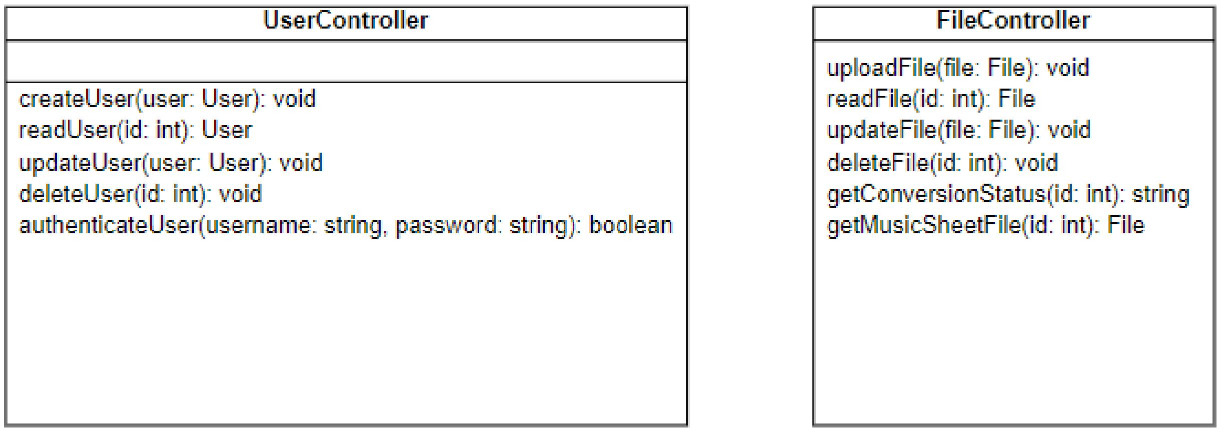
## 4.2 View Package

The components in the view package are responsible for displaying data to the user in an understandable and user-friendly format. The components also collect user input and provide feedback such as error messages and update the model package components as necessary.



## 4.3 Controller Package

The components in the controller package are responsible for receiving the collected user input from the view, processing the input and updating the models. The user controller handles the user authentication and authorization, by verifying the user credentials according to the requests received from the view package. It then updates the model package and prompts the view package for display accordingly. The file controller handles operations related to the saved music sheet, such as creating, editing, or deleting saved files.


UserController
createUser(user: User): void
readUser(id: int): User
updateUser(user: User): void
deleteUser(id: int): void
authenticateUser(username: string, password: string): boolean
FileController
uploadFile(file: File): void
readFile(id: int): File
updateFile(file: File): void
deleteFile(id: int): void
getConversionStatus(id: int): string
getMusicSheetFile(id: int): File

## 5. Glossary

- Pianissimo (noun): a passage marked to be performed very softly.
- MIDI: Musical Instrument Digital Interface
- PDF: Portable Document Format
- BPM: Beats per Minute
- FLAC: Free Lossless Audio Codec
- WAV: Waveform Audio File Format
- MP3: MPEG-1 Audio Layer 3
- AAC: Advanced Audio Coding
- GUI: Graphical User Interface
- RAM: Random Access Memory
- SOLID: Single responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle, and Dependency inversion principle
- OS: Operating System
- MB: Megabyte
- Kbps: Kilobits per Second
- IEEE: pronounced "Eye-triple-E," stands for the Institute of Electrical and Electronics Engineers. The organization is chartered under this name and it is the full legal name
- UML: The Unified Modeling Language (UML) is a general-purpose, developmental modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.
- REST: Representational State Transfer
- HTTP: Hyper-Text Transfer Protocol
- PY: Python
- QR Code: Quick Response Code
- GDPR: General Data Privacy Regulation
- MongoDB: This is a non-relational document database that provides support for JSON-like storage. The MongoDB database has a flexible data model that enables you to store unstructured data, and it provides full indexing support, and replication with rich and intuitive APIs.
- API: Application Programming Interface

- DB: Database
- HTTP: Hypertext Transfer Protocol
- MVP: Model-View-Presenter
- OS: Operating System
- POI: Point of Interest
- TCP: Transmission Control Protocol

## 6. References

- ACM Code of Ethics and Professional Conduct
- *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.
- “Code of Ethics.” Code of Ethics | National Society of Professional Engineers, July 2018, www.nspe.org/resources/ethics/code-ethics. Date Accessed: 14 October 2018.
- Jaggavarapu, Manoj. “Presentation Patterns : MVC, MVP, PM, MVVM.” Manoj Jaggavarapu WordPress, 2 May 2012, https://manojjaggavarapu.wordpress.com/2012/05/02/presentation-patterns-mvc-mvp-pm-mvvm/. Date Accessed: 25 December 2018.e
- “UML 2 Class Diagramming Guidelines.” Agile Design, agilemodeling.com/style/classDiagram.htm. Date Accessed: 18 February 2019.
- What Is UML.” *What Is Unified Modeling Language (UML)?*, www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 04- Oct-2020]
- “Bcrypt,” *PyPI*. [Online]. Available: https://pypi.org/project/bcrypt/. [Accessed: 21-Mar-2023].
- “Db.py,” *PyPI*. [Online]. Available: https://pypi.org/project/db.py/. [Accessed: 21-Mar-2023].
- “Librosa,” *librosa*. [Online]. Available: https://librosa.org/doc/latest/index.html. [Accessed: 21-Mar-2023].
- “OS - miscellaneous operating system interfaces,” *Python documentation*. [Online]. Available: https://docs.python.org/3/library/os.html. [Accessed: 21-Mar-2023].

- “Pandas documentation,” *pandas documentation - pandas 1.5.3 documentation*. [Online]. Available: https://pandas.pydata.org/docs/. [Accessed: 21-Mar-2023].
- “Pymongo 4.3.3 documentation,” *PyMongo 4.3.3 Documentation - PyMongo 4.3.3 documentation*. [Online]. Available: https://pymongo.readthedocs.io/en/stable/. [Accessed: 21-Mar-2023].
- “Python enhancement proposals,” *PEP 8 – Style Guide for Python Code*. [Online]. Available: https://peps.python.org/pep-0008/#class-names. [Accessed: 21-Mar-2023].
- “Python-dotenv,” *PyPI*. [Online]. Available: https://pypi.org/project/python-dotenv/. [Accessed: 21-Mar-2023].
- “RE - regular expression operations,” *Python documentation*. [Online]. Available: https://docs.python.org/3/library/re.html. [Accessed: 21-Mar-2023].
- “Tkinter - Python interface to TCL/TK,” *Python documentation*. [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed: 21-Mar-2023].

- Ohio State University Department Of Computer Science and Engineering, Systems Requirements Specification (SRS), 2006

- http://www.cs.fsu.edu/~myers/cop3331/notes/sysdesign.html

- https://thedotnetguide.com/mvc-design-pattern