Final Report

Team 2

May 11, 2023

1 Introduction

The intended user-base of our application, MyAudioEmailr, is professionals (e.g. senior executives), who typically receive large amounts of emails. In a professional context, staying apprised of emails is essential, because not doing so may result in missing important information. Keeping up with emails becomes more difficult the more emails there are to read. In a study of one software development company, Meyer et al. (2017: 1185) found that employees spent 14.5% of their time reading emails. This means that, in terms of working time spent on tasks, reading emails was the second-ranked task, after developing (Meyer et al. 2017: 1188). Employees also reported feeling that time spent on emails was unproductive (Meyer et al. 2017). These findings suggest that email plays a large role in professional life, and too much time spent on email may negatively affect productivity. With these aspects in mind, it seems that traditional email clients, which rely solely on text-based communication, may not fully meet the needs of professionals.

To meet the needs of professionals, MyAudioEmailr uses Text-to-Speech (TTS). Our application includes the functionalities common to email clients, meaning users can view their inbox, read, write and send emails. However, MyAudioEmailr can also read emails to the user, and the user can adjust the settings of speech, including speed, accent and scheduled reading time for their convenience. MyAudioEmailr is a web-based application. With this application, users can easily stay up to date with their emails, without having to review them manually. The goal of this application is to help professionals and other user types to save time, and help them to prioritise their work and lives more effectively.

2 Project scope and objectives

The application aims to integrate text-to-speech technology into an email client, providing users with a convenient and easy way to interact with their emails, especially when they cannot or should not look at the screen, such as while driving or exercising. Additionally, MyAudioEmailr aims to provide high-quality and natural-sounding voice output by incorporating Google TTS API. The email interface is designed to be clean and intuitive for browsing and managing emails. The use of MySQL as the primary database system ensures the security and efficient storage and management of email data.

The project scope for MyAudioEmailr is to develop a user-friendly web application that integrates TTS technology into an email client. The application will allow users to access and manage their email accounts using both text and voice input and output. The application will have two main parts: email and TTS. The email part will be developed primarily in Java on the spec- and front-end, and MySQL on the back-end. The TTS part will be developed primarily in Python.

3 Team profile

1. Georgia Gaffney; ggaffney2@sheffield.ac.uk

2. Nana Kayamori; nkayamori1@sheffield.ac.uk

3. Hok Yan Pun; hypun1@sheffield.ac.uk

4. Baiyang Qu; Bqu5@sheffield.ac.uk

5. Ziyu Wang; ZWang386@sheffield.ac.uk

6. Mingqing Zhang; mzhang122@sheffield.ac.uk

About the Gitlab: The branch that ultimately saves all the code is the project branch, not the default doc branch.
Link to the video: Video
Link to the team Google drive: Google Document
We have made our Google drive accessible to all staff at the University of Sheffield. However, please contact one of our team if there are any problems gaining access.

4 Product backlog

Backlog – A list of user stories that make up the complete product The following are our user stories, as defined in our user stories map (link is provided in Appendix). The user stories are given in descending order of importance. The importance was determined by the priority of the requirements according to our client. "As a user I want... :

- Highest importance / core functionality:
  - to access my account(s) using my email address(es) and password(s)
  - to write and send emails to other accounts
  - to receive emails from other accounts
  - my incoming and outgoing emails to be automatically sorted into different folders - 'inbox' and 'sent'
  - to be able to explicitly choose an email to read
  - to be able to explicitly choose an email to be read aloud
- Medium importance:
  - to see my inbox immediately once I log in
  - to choose multiple emails to be read, one after the other
  - to choose voices from a list that is varied by gender and accent
  - to have all unread emails automatically read to me
  - to have emails read to me by a specific time
  - to be able to change the speed of dictation, in order to reduce the time needed to listen to emails
  - to be able to have more than one email account/address
  - to send/receive emails to/from different email providers
- Low importance:
  - spam emails to be automatically sorted into a dedicated 'spam' folder
  - to be able to attach files to emails that I send
  - to change the apparent emotion of a voice
  - emails that have been written in Mandarin to be read in Mandarin

5 Analysis & Design

This part includes: System architecture, UML diagrams, Algorithm/Database design. The project architecture is made up of two main parts: TTS and email functionality.

5.1 Use case diagram

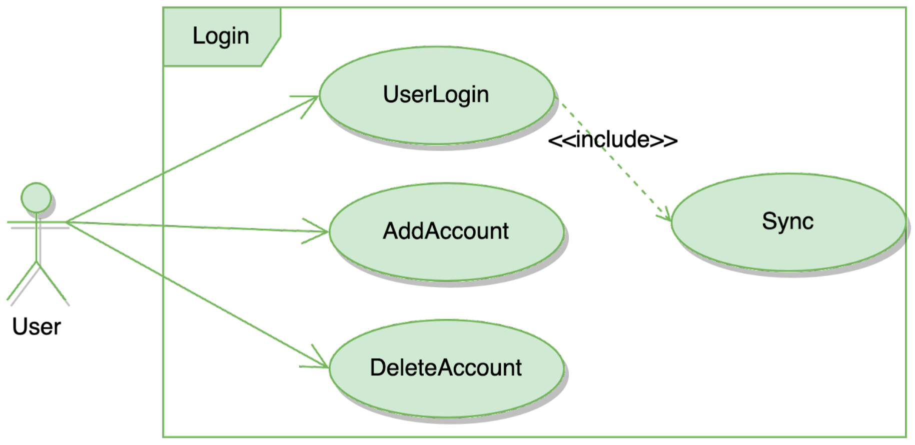


Figure1: UseCase Login

You can view other class diagrams in the appendix.

5.2 Sequence Diagram

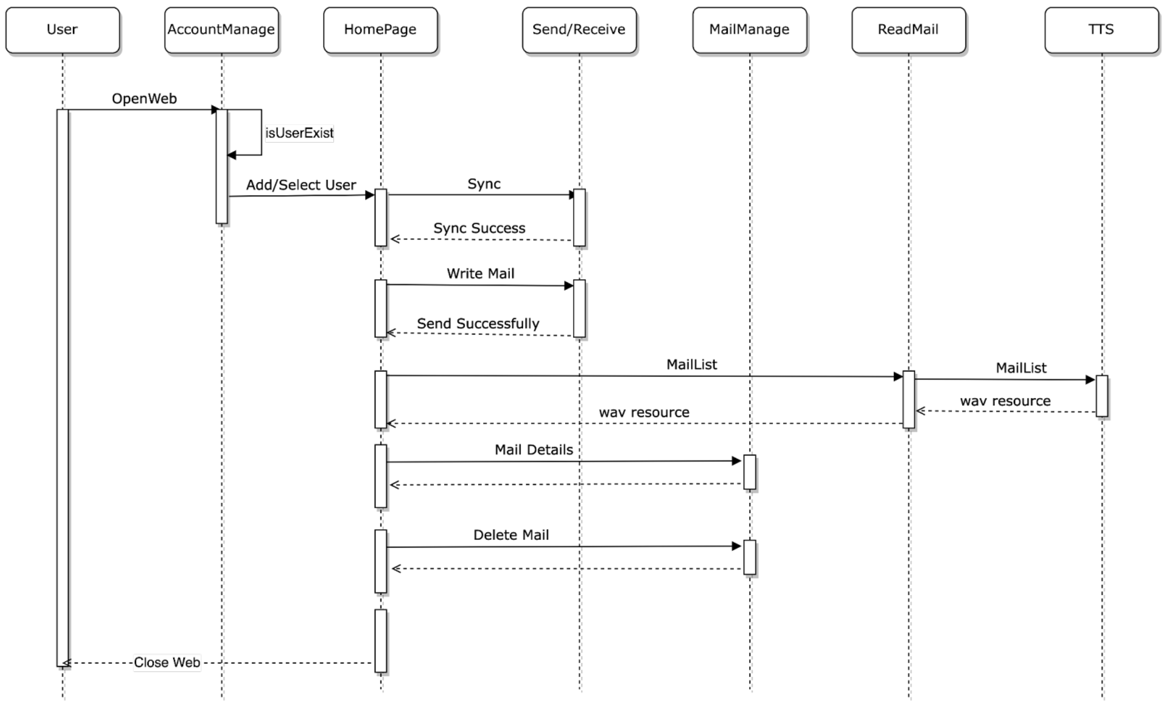


Figure2: SequenceDiag

5.3 Text-to-Speech (TTS)

The strategy of TTS we ultimately chose was an adaptation of Google's text-to-speech API. Initially, we used concatenative speech synthesis. Concatenative speech synthesis uses a database of speech sounds, which are overlapped and added together to form words. We chose this strategy as it has historically been one of the most widely used TTS methods in the industry, and thus there are several reference texts concerning concatenative TTS (Jurafsky & Martin, 2009).

Although the large amount of available information on concatenation meant we could be well informed on the implementation of concatenative TTS, we encountered some issues with this strategy.

Firstly, the resulting speech sounded inhuman, and the client raised this as a problem in the Week 8 meeting. The poor quality of the voice was due to discontinuities between diphones. Concatenative speech synthesis generally involves post-processing techniques to ensure the volume and pitch of the concatenated speech is relatively continuous, to the extent that joins are not perceptible (Jurafsky & Martin, 2009). However, due to the time constraints of the project, we were unable to implement these techniques. Secondly, Requirement 5 stated: "The user shall have the ability to select the preferred voice from a list of 2-3". When using concatenative speech synthesis, different voices are supported by recording different speakers pronouncing most of the diphones present in a language. However, we were only able to find a diphone dataset for one speaker.

Therefore, in order to address both the requirements and the client's concerns, we needed to choose a different strategy for synthesis. We chose the Python library gTTS, which is an interface of Google's TTS API. Not only did this library provide improved quality, but it also offered multiple voices. The support of different voices allowed us to address Requirement 5. In fact, gTTS offers 7 accents for English, which we enabled in our application, as we erred on the side of giving the user more options of accents, not fewer.

We wrote two scripts in Python, using PyCharm, to handle the TTS functionality. The first is `text_preprocessing.py`, which processes the email content and returns text. The second is `TTS.py`, which creates audio data using the external library gTTS and then processes the audio data. `TTS.py` is called by `httpHandler.py` (Section 6.2.7) and returns a path to a wav file.

5.3.1 Text Processing

This script takes an HTML file and turns it into a list of pronounceable words to pass to speech synthesis. There are 7 steps to process the text, hence 7 corresponding functions plus 1 pipeline function that calls all of them:

- `html2txt()`: takes an HTML file as the parameter, and turns it into plain text for processing using the Beautiful Soup module.
- `tokenize()`: breaks a text down to smaller, easier-to-process units. It takes a string; tokenizes the whole string using the `NLTK` library; and returns the whole text as a list of tokens.
- `strip()`: strips away useless parts from the tokenized text. It takes a list of tokens; removes punctuations that will not be converted into word representations for reading or used as markers in speech synthesis downstream; turns all remaining tokens lowercase; and returns a new complete list of tokens.
- `token_parser()`: breaks down a token that is a concatenation of smaller tokens. It takes a list of tokens returned from the previous function; parses from beginning to end; keeps punctuation marks that will be used as markers in speech synthesis downstream; extracts alphabetical chunks and/or punctuations to be read along the scan; and concatenates "'s" to its preceding token as it is meant in the original message. The function then returns the list of tokens.
- `math_parser()`: turns certain punctuation marks that were turned into word representations previously into a different word representation within a mathematical context. It takes a list within the list of lists; turns any "dash" or "dot" before a numeral into "minus" and "point" respectively; turns "asterix" between numerals into "times", and "sir cum flex" (dictionary-pronounceable representation of "circumflex") into "to the power of"; and concatenates numerals around any comma digit-separator. The function then returns the list of tokens with accurately (albeit not exhaustively) represented mathematical symbols if any.
- `num_parser()`: turns numerals into their word representations. It takes a list within the list of lists; turns any sequence of number and ordinal suffix into an ordinal number, and all numbers into their word representations.

Although the gTTS class implements text processing before synthesising speech, we found a number of benefits to using our own text processing. Benefits of using custom pre-processing include:

1. Being able to parse html, using the Python library 'Beautiful Soup', which was necessary for the email-specific TTS task of MyAudioEmailr.

2. Emails may contain textual elements such as links, and gTTS' pre-processor occasionally leads to extensions such as '.ac' being pronounced verbatim rather than spelled out. Due to our custom pre-processing, our synthesised speech spells out such extensions letter-by-letter, aiding listener understanding of links, internet addresses etc., which is essential for our demographic of professionals. As an example, gTTS' text processing and our text processing system would handle the internet address "sheffield.ac.uk/" differently. While gTTS would pass the address directly to the synthesiser, our text processing first tokenises such strings, for example, the internet address would become: "sheffield dot ac dot uk slash". Out-of-vocabulary/unusual tokens, such as 'ac', could then be pronounced letter-by-letter. In summary, using custom text-preprocessing allowed us to address issues not addressed by gTTS' text-preprocessing.

3. Being able to retain exclamation marks for increased volume in the speech synthesis stage. This helps to better convey tone in emails. Therefore, the overall TTS system of our application could be better suited to email synthesis, specifically.

5.3.2 Prosody and Synthesis

The `tts` function in the `TTS.py` script is primarily responsible for using the gTTS library to synthesise phrases, and then concatenating them, to create a complete utterance (i.e. a complete dictation of an email). It takes as parameters: wav_name: a string that will be the name of the resulting wav file; email_content: the HTML content of a given email as a string; accent, a string; and speed, a string, with the default of 'normal'. The following is an example usage:

```
speech = tts(wav_name='testus',
             email_content="""<p>Hello world!</p>""",
             accent='us',
             speed='fast')
```

There were two main reasons for using a single function for synthesis. Firstly, our application required that the synthesised speech be passed to the email part. If we were to use several functions to synthesise speech, we would need a 'main' function, that would be called by the email part, and call each of `TTS'` functions in turn. However, this may have slowed down processing, due to repeated function calls. A single function does not raise the same issues, as there is thus only one function call to pass speech to the email part. Secondly, it was necessary to use a `try... except...` structure to prevent the code from attempting to create empty speech objects (more information on this specific problem, and how we addressed it, is given below in this same section). In this instance, it was simpler to use the `try... except...` structure with a single function, as it meant that either speech or an error code would be returned to the email part. With these two reasons in mind, we decided that a single function should synthesise and return speech to the email part. It will now be explained why we used the gTTS library.

We have three main reasons for using gTTS:

1. gTTS has comparably fewer dependencies than other Python libraries that perform TTS, such as 'coqui-ai TTS';

2. it has wider platform support than another Python TTS library 'espeak-ng' (e.g. it can run easily on Apple OS);

3. finally, gTTS allows for speech data to be returned as an object, unlike libraries such as 'pyttsx3'. As speech can be captured in a variable as byte data, gTTS enables post-processing of speech. We were able to take advantage of this by processing speech on the phrase-level, rather than on the entire email contents simultaneously. More detail on what phrases are in this context, and how `tts` processes them, are given below.

A phrase, in the context of our application, is defined as text between certain delimiters, which are: ".,!?". For example, given the HTML input string:

```
"""<p>The sun was setting; the sky was painted in shades of
orange, pink, and purple.</p>"""
```

The phrases would be:

```
the sun was setting the sky was painted in shades of orange ,
 pink ,
 and purple .
```

Each phrase is shown on a new line, having been split by the delimiters. Synthesising the speech on a phrase-by-phrase basis allowed for more control of the output than would be allowed by synthesising the entire email contents at once. For example, after converting byte data to float64 arrays, it is possible to increase the volume of phrases ending in an exclamation mark, by simply multiplying the array elements by 2. However, there are a number of steps executed by the `tts` function, before synthesising the phrases.

The first step executed by `tts` is to check whether the user input for the accent parameter is supported. This check is achieved by using a list containing all of the supported accents as strings. If the input is not in this list, an exception is raised and the function is exited. `TTS.py` imports `text_preprocessing.py`, meaning that it can call the pipeline `text_proc` function to process the email contents into a form that can be synthesised.

Next, a `try... except...` block is opened. This structure is necessary because there are phrases which contain no alphanumeric characters. Passing such a phrase to the gTTS class while break the code and throw `AssertionError: No text to send to TTS API`. However, this error is not visible to the user on the front-end. Without a check, nothing would be played nor shown to the user, with no explanation. Although we implement text-preprocessing, accounting for all possible characters is not feasible, and thus we need a catch on the synthesis end, in case any non-synthesizable strings elude text-preprocessing. Therefore, if such strings are encountered during synthesis, our first method to avoid this problem is to return an error code -1 to `httpHandler.py`. This error code can then be passed to the email part so that an error can be shown to the user. This is more meaningful to the user than no sound playing, with no explanation given.

The second method is implemented within the `for` loop that is called on each phrase in the processed text. Speech data for each phrase are collected as arrays in a list. Once this `for` loop is entered, there is an `if... else...` structure that appends an array of zeros to the list in place of speech data, if the phrase has no alphanumeric characters. Such a phrase can thus be pronounced as silence. In such cases, the sampling rate, which is declared as variable `fs` above the `for` loop, is set as 24,000 Hz, to make it compatible with the gTTS speech data of the same sampling rate. If a phrase does contain alphanumeric characters, then it is handled by the `else` block. The phrase and selected accent are passed to the gTTS class, and the resulting speech data are converted from bytes to float64 encoding through the use of a buffer and the Soundfile library, which also returns the sampling rate. The speech data are then trimmed of silence to varying degrees, depending on punctuation. For example, the trailing silence of a phrase ending in a full-stop is left longer than that of a phrase ending in a comma. Once all arrays have been appended to the list, it is concatenated as a single array. The speed of this array can then be shifted up or down, if the user has selected a speed change. This is achieved by either increasing or decreasing the sampling rate, respectively. Soundfile is used to write all speech data for a given email to a wav file. Finally, a path to the created wav is returned when `tts` is called by `httpHandler.py`.

6 Email Functionality

6.1 Technology selection

- Backend: SpringBoot, SpringShell
- Frontend: Thymeleaf, Bootstrap, JavaScript
- Database: MySQL
- Mail Framework: JavaMail

The reasons we chose these tools are as follows:

We chose SpringBoot as the backend technology because of its ease of use and development speed. SpringBoot provides a range of pre-configured features that simplifies the setup of new projects. It also has a large and active community that provides support and frequent updates, making it a reliable choice for building scalable and maintainable web applications.

MySQL was chosen as the database because it is open-source, scalable, and reliable. It is widely used and has a large community that provides support and resources, making it a safe and well-established choice for data storage.

JavaMail is a Java API used to send and receive email via SMTP, POP3, and IMAP. It provides a platform-independent and protocol-independent framework to build Java-based email client applications. JavaMail includes classes for handling attachments, forwarding, reply, HTML messages etc. It is widely used in enterprise applications and provides a flexible and powerful solution for email communication.

Lastly, we use SpringShell so that `TTS` developers do not need to perform additional configurations. Developers only need to install `Java JDK` and run the `Jar` file directly through the command line to conduct integration testing in the early stages.

Overall, our tool selections were made based on their ease of use, flexibility, reliability, and community support, among other factors. Information on Thymeleaf and Bootstrap Will be given in Section 6.2.8, as such information concerns the frontend.

6.2 Function Details

6.2.1 Login

When a user logs into the system, the system checks if a `deviceId` is stored in the cookie. It is a string composed of "current time - 6 random letters" that is almost impossible to duplicate. If the `deviceId` is detected, the system will query the accounts in the database for a match and display the `SelectAccount` page. If not, the login page is displayed.

Additionally, the system automatically sends a request to the `IMAPserver` on every login to obtain the latest emails and stores them in the system's database.

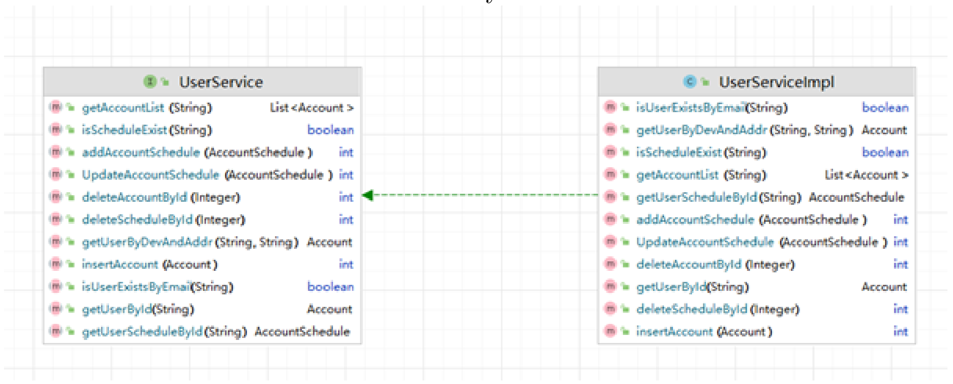

UserService
getAccountList (String) List<Account>
isScheduleExist (String) boolean
addAccountSchedule (AccountSchedule) int
UpdateAccountSchedule (AccountSchedule) int
deleteAccountById (Integer) int
deleteScheduleById (Integer) int
getUserByDevAndAddr (String, String) Account
insertAccount (Account) int
isUserExistsByEmail(String) boolean
getUserById(String) Account
getUserScheduleById (String) AccountSchedule
UserServiceImpl
isUserExistsByEmail(String) boolean
getUserByDevAndAddr (String, String) Account
isScheduleExist (String) boolean
getAccountList (String) List<Account>
getUserScheduleById (String) AccountSchedule
addAccountSchedule (AccountSchedule) int
UpdateAccountSchedule (AccountSchedule) int
deleteAccountById (Integer) int
getUserById(String) Account
deleteScheduleById (Integer) int
insertAccount (Account) int

Figure4: SequenceDiag

6.2.2 Reading mails

When a user requests to read a single email, the system will directly send the email to the TTS part and wait for the conversion to complete. This function is lacking, as once the function is triggered, the user cannot determine the current progress of the email conversion.

Both "reading some specific emails" and "reading all unread emails" functions use Socket. The logic for triggering these two functions by clicking buttons or through scheduled tasks is the same, the only difference being whether they are triggered by a timed task.

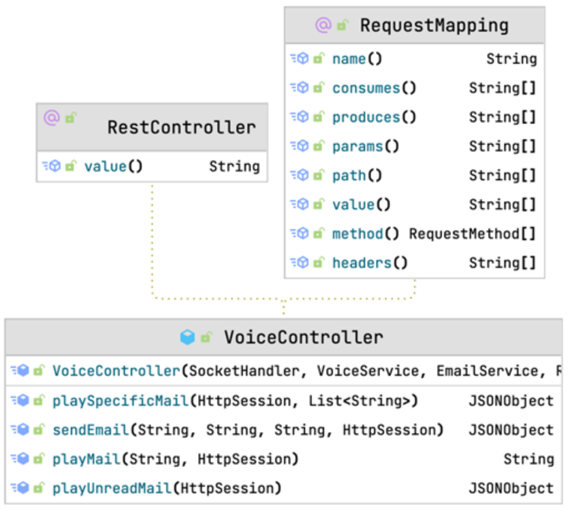

RequestMapping
name() String
consumes() String[]
produces() String[]
params() String[]
path() String[]
value() String[]
method() RequestMethod[]
headers() String[]
RestController
value() String
VoiceController
VoiceController(SocketHandler, VoiceService, EmailService, R
playSpecificMail(HttpSession, List<String>) JSONObject
sendEmail(String, String, String, HttpSession) JSONObject
playMail(String, HttpSession) String
playUnreadMail(HttpSession) JSONObject

Figure5: VoiceController

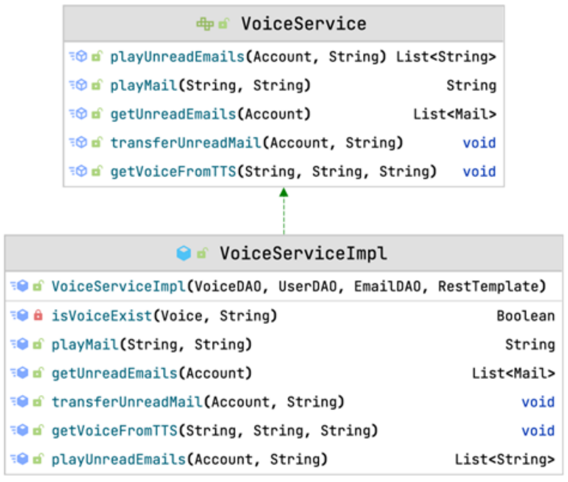

VoiceService
playUnreadEmails(Account, String) List<String>
playMail(String, String) String
getUnreadEmails(Account) List<Mail>
transferUnreadMail(Account, String) void
getVoiceFromTTS(String, String, String) void
VoiceServiceImpl
VoiceServiceImpl(VoiceDAO, UserDAO, EmailDAO, RestTemplate)
isVoiceExist(Voice, String) Boolean
playMail(String, String) String
getUnreadEmails(Account) List<Mail>
transferUnreadMail(Account, String) void
getVoiceFromTTS(String, String, String) void
playUnreadEmails(Account, String) List<String>

Figure6:VoiceService

6.2.3 Send & Receive Mails

Upon successful login, the user is automatically redirected to the inbox page, where they can easily view, delete, and read all the emails that are currently present in their inbox. The backend component of the system is responsible for synchronising email data every ten minutes, and users can also manually synchronise their email information by clicking on the "Sync" button on the top right of the page. By clicking on the "details" button, users can view email details, including email text.

When a user decides to read an email, the system sends a request to the TTS module, which then processes the request and generates an audio file based on the email content. The TTS module subsequently sends the audio file information to the email system, which stores it in the database and sends the email information (i.e., the generated file path) to the frontend via a socket. The frontend listens for this event and displays a pop-up window that allows the user to play the audio of the email content.

To send emails, the system uses the Simple Mail Transfer Protocol (SMTP), which can be done by clicking on the "write message" menu.

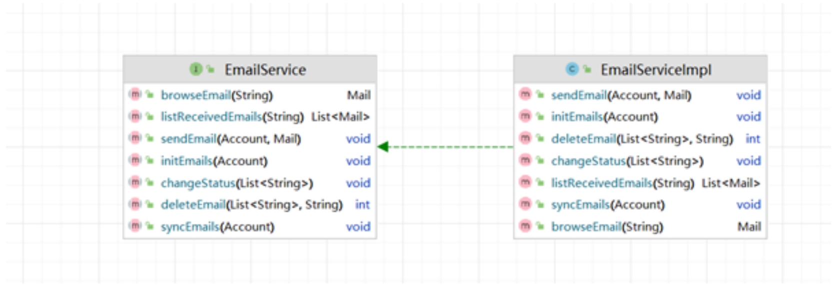

EmailService
browseEmail(String) Mail
listReceivedEmails(String) List<Mail>
sendEmail(Account, Mail) void
initEmails(Account) void
changeStatus(List<String>) void
deleteEmail(List<String>, String) int
syncEmails(Account) void
EmailServiceImpl
sendEmail(Account, Mail) void
initEmails(Account) void
deleteEmail(List<String>, String) int
changeStatus(List<String>) void
listReceivedEmails(String) List<Mail>
syncEmails(Account) void
browseEmail(String) Mail

Figure7: EmailService

6.2.4 SocketHandler

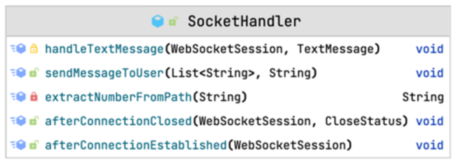

SocketHandler
handleTextMessage(WebSocketSession, TextMessage) void
sendMessageToUser(List<String>, String) void
extractNumberFromPath(String) String
afterConnectionClosed(WebSocketSession, CloseStatus) void
afterConnectionEstablished(WebSocketSession) void

Figure8: SocketHandler

`SocketHandler` handles socket requests. When users need to read multiple emails, each email converted to speech will be sent one by one in order. This way, we can avoid users waiting too long for emails to be read.

6.2.5 SMTP and IMAPHandler

These two classes are used to send requests to SMTP and IMAP servers. The IMAP server is used to receive emails. `IMAPHandler` converts Message and `MimeMessage` objects from the server to an `ArrayList<Mail>` type and returns them. `SMTPHandler` converts `Mail` type objects to `MimeMessage` type and sends them to the server.

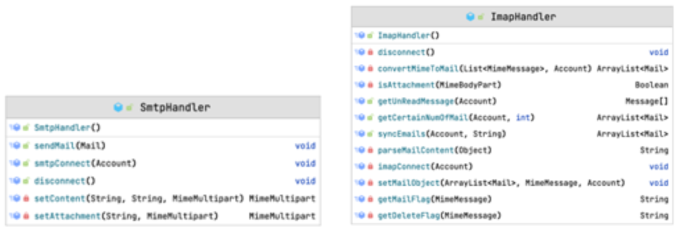

SmtpHandler
SmtpHandler()
sendMail(Mail) void
smtpConnect(Account) void
disconnect() void
setContent(String, String, MimeMultipart) MimeMultipart
setAttachment(String, MimeMultipart) MimeMultipart
ImapHandler
ImapHandler()
disconnect() void
convertMimeToMail(List<MimeMessage>, Account) ArrayList<Mail>
isAttachment(MimeBodyPart) Boolean
getUnReadMessage(Account) Message[]
getCertainNumOfMail(Account, int) ArrayList<Mail>
syncEmails(Account, String) ArrayList<Mail>
parseMailContent(Object) String
imapConnect(Account) void
setMailObject(ArrayList<Mail>, MimeMessage, Account) void
getMailFlag(MimeMessage) String
getDeleteFlag(MimeMessage) String

Figure9: SmtpHandler and ImapHandler

6.2.6 Settings

On the 'Settings' page, users can configure their account settings, such as the voice accent. MyAudioEmailr only supports English, for which there are the following accents: Australia, United Kingdom, United States, Canada, India, Ireland, and South Africa. Users can also set the time for scheduled email reading on this page. Once a scheduling is saved, the system will automatically convert and play the emails as speech at the selected time every day.

The scheduled tasks consist of reading all unread emails at a specific time, initiating the conversion of emails to speech at a particular time, and background checking for new emails. Since each task is user-specific, we cannot use the @Scheduled annotation in Spring Boot to automatically start them during development. Therefore, we implement Runnable to write the scheduled tasks and use cron expressions to set the trigger time.

To improve the efficiency and speed of email to speech conversion, the backend will start TTS one hour in advance of the scheduled time. This will ensure, in most cases, that the speech files are ready by the selected time.

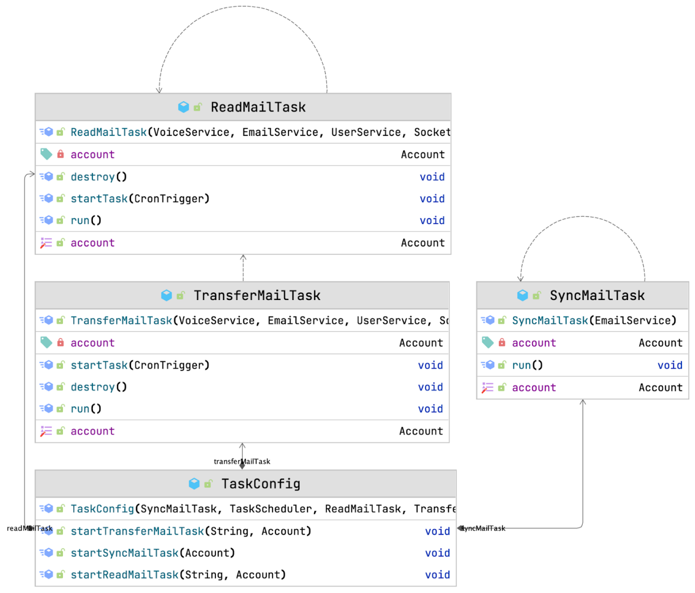

ReadMailTask
ReadMailTask(VoiceService, EmailService, UserService, Socket
account Account
destroy() void
startTask(CronTrigger) void
run() void
account Account
TransferMailTask
TransferMailTask(VoiceService, EmailService, UserService, So
account Account
startTask(CronTrigger) void
destroy() void
run() void
account Account
SyncMailTask
SyncMailTask(EmailService)
account Account
run() void
account Account
transferMailTask
TaskConfig
TaskConfig(SyncMailTask, TaskScheduler, ReadMailTask, Transfe
readMailTask
startTransferMailTask(String, Account) void
startSyncMailTask(Account) void
startReadMailTask(String, Account) void
syncMailTask

Figure10: scheduledTasks

6.2.7 TTS Model and TTS httpHandler

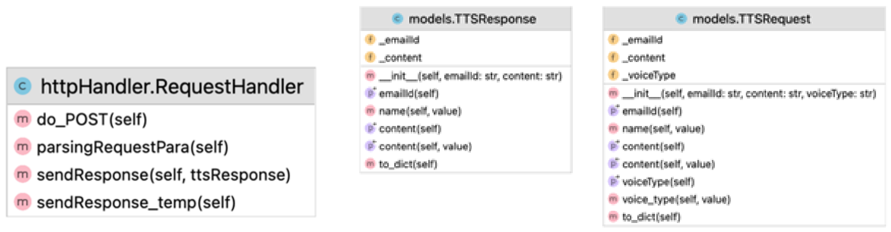


Figure11: httpHandler and TTSModel

We use the Python script httpHandler.py (pictured above, bottom image) to allow communication and transfer of data between our TTS part (written primarily in Python) and our email part (written primarily in Java). Communication and transfer are achieved through https requests. The script contains the function 'startServer' and the class 'RequestHandler', which inherits methods and behaviours from the 'BaseHTTPRequestHandler' class from the Python http.server library. The class within httpHandler.py, 'RequestHandler', also relies on two classes imported from the team-created models.py script, which are:

1. 'TTSRequest', for transferring data (e.g. email content) from the email part to the TTS part;
2. 'TTSResponse', for transferring data (e.g. the path to the synthesised speech) from the TTS part to the email part.

'RequestHandler' is executed when it is called by 'startServer'. The class handles http requests using 3 methods, detailed below.

`do_Post` checks the hosts path, and if the path is correct, it gets the email id, email contents, and voice settings from the email component. It then passes these data as parameters to the 'tts' function and gets back a path to a wav file containing the synthesised email contents. When conversion is complete, the email id and path to the wav file are sent to the email part, using 'TTSResponse'. If the path is not correct, then a 404 error message is thrown.

`parsingRequestPara` is called by `do_Post` in order to get and parse the initial request from the server, after the email part has made a request to the server. It reads the request while decoding it to utf-8. It then loads the request as a json, and then uses 'TTSRequest' in order to return data, which are the email id, email contents, and voice settings.

`sendResponse` is also called by `do_Post` and prepares the data from the `TTS` part to be transferred to the email component.

6.2.8 Frontend

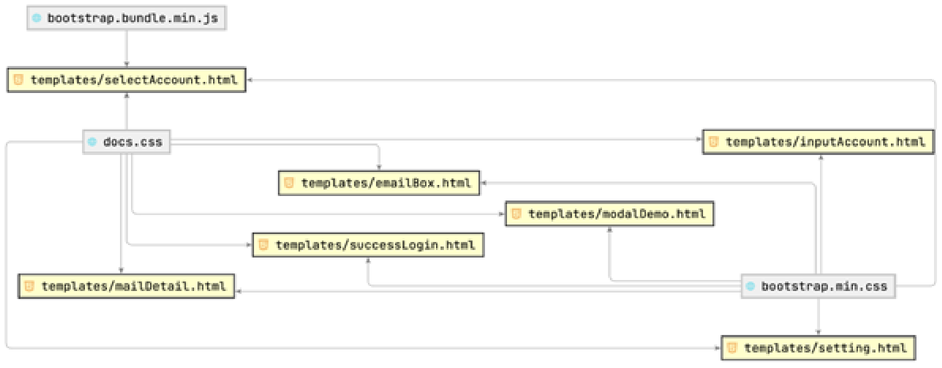


Figure12: FrontEnd

There are several reasons we chose Thymeleaf and Bootstrap:

Strong adaptability: Bootstrap is a popular front-end framework that provides many ready-made components and styles, making it easy to create responsive layouts that work on various devices. Thymeleaf is a popular server-side template engine that can easily render dynamic data into front-end pages.

High development efficiency: Bootstrap provides a large number of CSS and JavaScript components and plugins, enabling developers to quickly create attractive UI designs and reducing the time spent writing CSS and JavaScript. Thymeleaf is a server-side `Java` template engine that allows for seamless integration with SpringBoot. Additionally, Thymeleaf provides an intuitive way to define dynamic data and page rendering, allowing developers to focus more on implementing business logic.

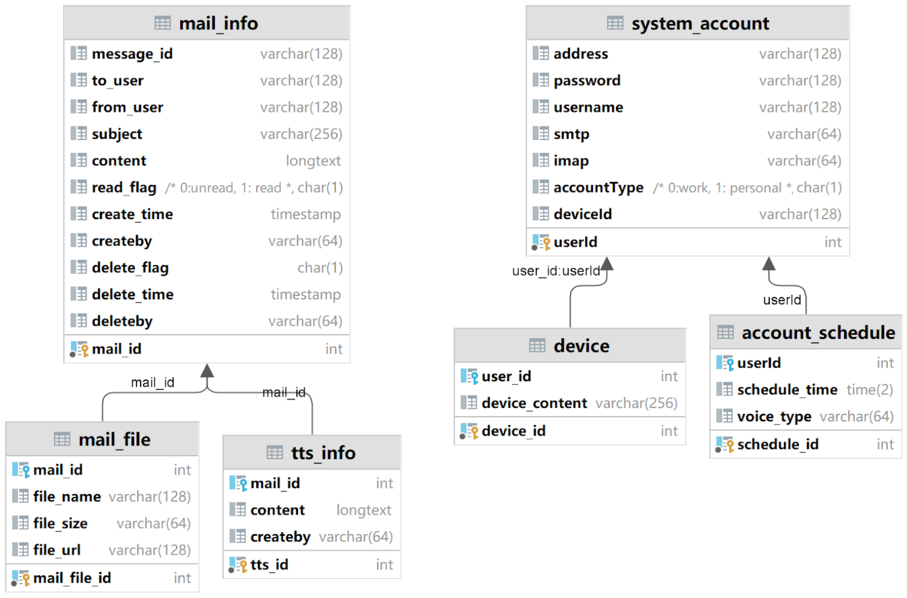
Easy to learn: Both Bootstrap and Thymeleaf have extensive documentation and community support, with many examples and tutorials available to make learning and using them easier. The syntax of Thymeleaf is easy to learn and understand, and it supports both HTML and XML markup, making it a flexible option for designing web pages.

Strong extensibility: Because Bootstrap and Thymeleaf are both open-source, they can be easily extended and customized to meet the needs of different projects.

In summary, using Thymeleaf and Bootstrap can improve development efficiency, reduce the time spent writing CSS and JavaScript, create more attractive and responsive UI designs, and be easy to learn and extend.

6.3 Database Diagram

The `audioEmail` database schema includes several tables for managing email and user accounts. The `mail_info` table stores information about emails, including message ID, sender, recipient, subject, content, read status, and timestamps for creation and deletion. The `mail_file` table stores information about attachments for emails, including the filename, size, and URL. The `system_account` table stores user account information, including the user ID, email address, password, username, and settings for SMTP, IMAP, account type, and device ID. The `device` table stores information about devices associated with user accounts, including the device ID, user ID, and device content. The `tts_info` table stores text-to-speech information, including the ID of the associated email and the content to be read. The `account_schedule` table stores information about scheduled text-to-speech conversions, including the ID of the associated user account, the scheduled time, and the voice type to be used. All tables include foreign keys to maintain referential integrity.



mail_info	
message_id	varchar(128)
to_user	varchar(128)
from_user	varchar(128)
subject	varchar(256)
content	longtext
read_flag	/* 0:unread, 1: read */, char(1)
create_time	timestamp
createby	varchar(64)
delete_flag	char(1)
delete_time	timestamp
deleteby	varchar(64)
mail_id	int

system_account	
address	varchar(128)
password	varchar(128)
username	varchar(128)
smtp	varchar(64)
imap	varchar(64)
accountType	/* 0:work, 1: personal */, char(1)
deviceId	varchar(128)
userId	int

device	
user_id	int
device_content	varchar(256)
device_id	int

account_schedule	
userId	int
schedule_time	time(2)
voice_type	varchar(64)
schedule_id	int

mail_file	
mail_id	int
file_name	varchar(128)
file_size	varchar(64)
file_url	varchar(128)
mail_file_id	int

tts_info	
mail_id	int
content	longtext
createby	varchar(64)
tts_id	int

Figure13: DiaDatabase

7 Evidence of Testing - Test plan, Test Documentation and Test Results

7.1 Test Tools

During the testing process, various tools can be used.

- SpringShell
- JupyterNotebook
- Python Script

At the begining, we considered using Postman for API testing and JUnit for white box testing of MailPart. However, we realised that using Postman would add an extra learning curve for team members since the inter-system call is not complex. Therefore, we decided not to use this tool.

At the same time, we encountered some strange problems while configuring JUnit. After running the test cases, we did not receive any response. After several days of trying, we were forced to abandon JUnit and use SpringShell as a temporary replacement.

Regarding TTS, testing was performed through the dedicated file `test.py`. This file imported the TTS module and from it called the tts function. The file could then be used to pass test strings. For example, in whitebox testing, we would pass different html strings that may pose varied problems (e.g. text containing foreign characters or unusual punctuation). By having unit testing specifically for the TTS part of our app, we saved time, as we did not run the entire application,

7.2 Examples of Test Cases

The commit record means: the commit id when the bug is fixed. These are just some of the bugs that took us a lot of time. For other bug fixes, check the commit log in gitlab. When doing the white box testing, most function did not have many `if` statements, so the coverage is not a key metric for us.

Test case 1: Read mail

- Test step:
  Input is ‘As an AI language model my knowledge cutoff is 2021’
- Expected Result:
  The sentence above is the input content, it should read ‘2021’ as ‘twenty twenty one’
- Test Result:
  It just reads ‘one’
- Commit record:
  42a672e95ab3409e4913e437d138111edec408de

Test Case 2: Read a single email

- Test step:
  After login, click the read button of an email
- Expected result:
  User should be able to see the audio dialog box
- Actual result:
  Nothing happened
- Commit record:
  da95b010d6c7e8566a1cc4735f9826c568bcc04e

Test case 3: Wav generation time test

- Test step:
  I sent an email that was 100 words long, to test the time taken to generate the wav file
- Expected result:
  The time taken should be under one second
- Actual result:

Content length (no. words)	Punctuation	Generation time (sec)	Wav length (sec)
40	NO	10	30
42	YES	12	32
100	NO	25	85

- Commit record:
  42a672e95ab3409e4913e437d138111edec408de

Test Case 4: Common French in email

- Test step:
  Added some common french words in an English sentence
- Expected result:
  Read all of the words or skip any unknown words
- Actual result:
  French letters caused TTS part to throw errors
- Commit record:
  42a672e95ab3409e4913e437d138111edec408de

More test cases can be found in the Appendix.

8 Team management & communication

Organisation of the team and use of tools:
Each of us is an active team member and project builder, however, we also adopted specific roles. For example, while Mingqing, Nana and Ziyu primarily worked on the email part, Baiyang, Georgia and Hok Yan primarily worked on the TTS part. We spent the first week organising the team and defining the roles and responsibilities of each member before we started the technical part of the project. We set reminders for each important time point and then planned all tasks for each phase. Our model for carrying out the project was to meet twice a week to plan and summarise, and to discuss and resolve any issues we were encountering. Each member allocated several hours outside of the meetings to work on programming.

In the second week, once we met our team members, we wrote our team operating agreement. This document defined how to manage our team and communicate for the rest of the project. The latest version of this document is available from our Google Drive, the link to which is in the Appendix. We assigned specific members to work on each task (e.g. on Trello). However, members were also free to assist on a task they were not assigned to.

How the team communicated throughout the project:
We held bi-weekly meetings, which were attended by all members. In the few cases where not all members were able to meet in person, members were able to attend remotely using Google Meet. These bi-weekly meetings were the Tuesday lab session, and an additional team meeting every Thursday. The main purpose of the Thursday meeting was to pair-program, resolve any issues that team members had encountered and push code to GitLab. In preparation for the client meetings, we held additional meetings on Monday evenings in Weeks 8 and 11.

During meetings, Georgia acted as facilitator and checked everyone's progress. We then defined our respective tasks before the next meeting. We kept minutes of the meetings, an excerpt of which is presented in the communication section. Outside of meetings, we used a group chat on the online messaging platform 'Whatsapp' to provide support in a timely manner and give each other quick updates. We also used this chat to vote on decisions, using the Whatsapp 'poll' function. For example, we used a Whatsapp poll to vote on whether we should take a break over the Spring holiday or we should treat the Spring holiday as another sprint. We voted unanimously in favour of the latter. We used Miro for user stories and Trello for backlog to track tasks and progress.

All code and README files were shared via Gitlab. We also had a Google drive specific to the team project for formal communications: meeting agendas and minutes; specifications; and notes of issues to be raised with the advisor.

During the course of the project, members who were more familiar with a given area would support other members. For example, Mingqing created and shared JSON.md and JSON.pdf to aid his teammates' understanding of `json` objects and how they can be used to transfer data. These files can be accessed on our Google drive.

8.1 Management

- Team Collaboration
  - Georgia is responsible for team coordination and communication, moderates meetings, and organises documents.
  - Hok Yan Pun is responsible for arranging meeting locations and editing documents.
- System Design
  - UI and function Design
    - All team members participated in this part.
  - Research on SMTP and IMAP (Nana)
  - Research on TTS API (Baiyang)
  - Research on hashing password (Georgia)
  - Database Design (Ziyu; Baiyang)
- System Development
  - TTS
    - Text processing (Hok Yan)
    - Speech synthesis (Georgia)
    - Connector to Email (Georgia; Mingqing)
  - Mail Part
    - Backend
      - Login (Ziyu; Mingiqng)
      - Scheduled Task (Mingiqng; Ziyu)
      - Send/Receive Mail (Mingqing; Ziyu; Baiyang)
      - Handlers of IMAP/SMTP (Nana; Mingqing)
      - Settings (Ziyu; Mingqing)
      - Connector to TTS (Mingqing; Georgia)
    - Frontend
      - Login (Ziyu; Hok Yan; Nana)
      - Inbox (Ziyu; Nana; Hok Yan)
      - Setting (Ziyu; Mingqing; Hok Yan)
- System Testing
  - White Box Test
    - TTS (Georgia)

        * Email (Baiyang)
    - Black Box Test
        * Email (Nana)
    - Bug Fix

- Final Report (All team members participated in this part.)

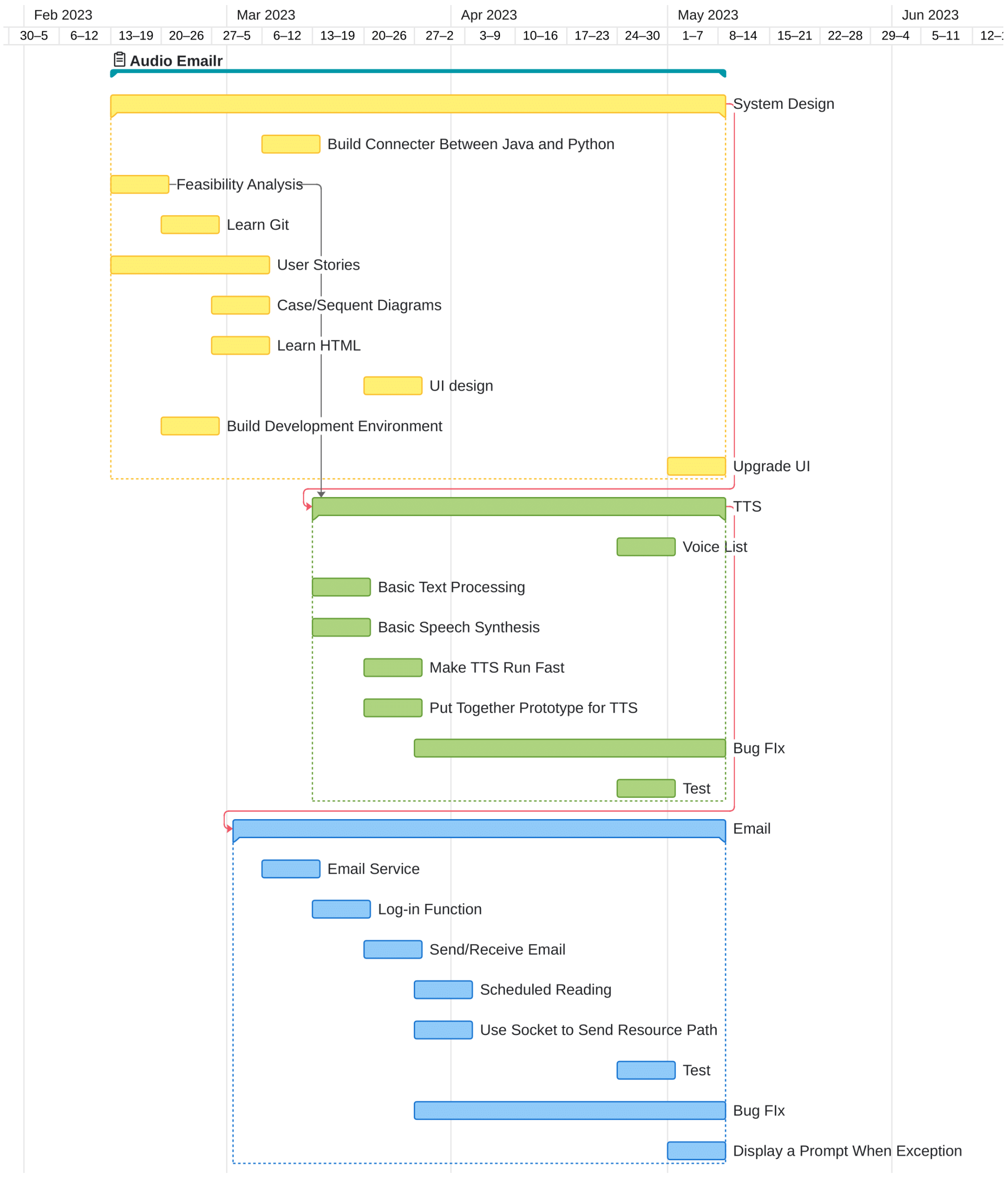


Figure14: GanttChart

The above is the overall task management of the project, which was mostly unchanged from the beginning of the project. Mingqing, Ziyu and Nana focused on the email functionality from the start, and Georgia, Baiyang and Hok Yan on TTS. As certain sub-tasks were completed, Baiyang joined the email back-end, and Nana and Hok Yan moved onto the front-end; while Georgia was responsible for implementing the new speech synthesis strategy, and Mingqing and Ziyu remained the pillars of the email back-end. As some members were not familiar with git, certain functionalities were developed, tested and debugged mostly on local drives, with occasional commits to GitLab that included multiple fixes.

Minutes usually followed this structure:

- each member summarised what they had been working on in the past week;
- we would take note of any reminders to ourselves that resulted from our discussions;
- each member updated the team on what they planned to work on next. This helped us to avoid multiple people working on the same task, and thus saved us time.

There have been no major disagreements in the team. When issues arose, we discussed together as a team, considering the practicalities of the project; each member's abilities; and considered the client requirements to reach consensus. If discussion did not reach consensus, we used polls, as mentioned above. In the case of miscommunication, we all tried to be understanding and just focused on solving the problem at hand.

9 Planned & Completed Features

This section presents the user stories that were planned and completed in each sprint. All of the features listed in PROJECT REQUIREMENT FORM are completed. We have also completed a number of extra functions to improve user experience.

- Week2
  - List user stories
  - Feasibility analysis (SMTP/SPAM)
- Week3
  - Learn git
  - Build development environment
  - UI Design
- Week4
  - Use case diagrams
  - Sequence diagrams
  - User story mapping
  - The priority of the user stories
  - How to handle HTML
- Week5
  - Build connector between Java and Python
  - EmailService
- Week6
  - Check if we have the basic functionality ready, and then decide which non-essential features to work on first
  - Login functionality
  - Basic text processing
  - Basic speech synthesis

- Week7
  - Send and receive mails
  - Research how to speed up TTS
  - Put together prototype for TTS
- Week8
  - Bug fix
  - Scheduled Task
  - Use socket to send resource path
- Week9
  - Voice list
  - System Test
- Week10
  - Bug fix
  - Upgrade UI
- Week11
  - Display a prompt when the software throws an exception

10 Extra Completed Features

- More voice options
  The requirement asked for 2-3 kinds of voice, we offer 7 different accents to provide greater choice.
- Account delete
  We added an account deletion button on the login page to delete accounts that are no longer needed.
- Sync function
  On the inbox page, there is a sync button in the top right corner that can be used to sync emails received in real time. Syncing also be uploads data to our database.
- Emails delete
  Users can indicate one or more emails to delete on the inbox page.

11 Uncompleted Features

This section is about the features that were not completed and the reasons why.

- Password hashing
  Initially we hashed the user's password, but then we discovered that the App password used by the IMAP protocol had to be a cleartext password when we fetched the user's email message via IMAP.
- Clearly distinguish between read and unread.
  The project uses an additional column to the email list to differentiate the read/unread status of the emails. This approach may not be as visually clear as other email clients (e.g. Google Mail uses different background colours to distinguish read/unread). The reason for this is that we lacked sufficient time to research design.
- Display conversion progress
  There is no indication of text-to-speech conversion under progress, so the user would not know what is happening with the program if the conversion takes too long. This problem was not foreseen in the design stage.

12 Conclusion

12.1 What we have learned

The following summarises the knowledge we have gained through this project:

The importance of teamwork: We were able to successfully complete the project through communication, co-ordination and support. We learnt how to collaborate as a team, how to distribute tasks, and how best to support others in the team.

Technical skills enhancement: In this project we used a variety of techniques and tools, such as productivity managers (e.g. Trello) and software libraries/engines (e.g. NLTK, Thymeleaf). This has expanded our skill-set and sharpened our problem-solving skill. We have also learnt how to write better, more readable code and how to conduct code reviews.

The importance of project management and planning: We learned how to set clear objectives and deadlines, as well as how to track progress and manage risks. We learned to change tactic if required by the client. For example, we changed the speech synthesis technique in order for the speech to have improved quality.

12.2 Challenges we have faced and resolved

We also encountered some challenges in this projects. One of these issues was the diversified team background. Everyone in the team has a different academic background. For example, two members had an undergraduate background in linguistics and one in Electronic Information Engineering. Among the other three members, one member had more experience with C++ but lacked web development experience, while the other two had more Java development experience.

Our corresponding solution was:

To maximise the development efficiency, we decided that three members (Baiyang, Georgia and Hok Yan) would be responsible for TTS based on Python, as they were taking the Speech and Language Processing program, and they all had a foundation in Python. The three remaining members, on the Advanced Computer Science program would be responsible for web development and design. Since these members did not all take the IntelligentWeb course and were not particularly proficient in Javascript and Nodejs, we ultimately decided to use Java for the development. Additionally, during the system design and requirements analysis phase, this diverse background provided us with great help, and each person put forward valuable opinions on the system's functionality and design.

Another problem was that we initially underestimated the complexity of the project. We believed that three people each could develop the TTS and email parts, respectively, and still meet our schedule. As a result, in the early stages of the project, only one or two members built the entire Java and Python development environment, independently of one another. Our underestimation of the project's complexity also led to the fact that we did not set up a clear and specific testing plan. Therefore, it was not until later in the development process that we started to spend more time on testing. Additionally, during the initial testing process, we relied on the more experienced members of the team to test the features they developed. However, developers may have a bias towards their own code and may overlook certain defects or issues. Having separate testers would have ensured that there was an unbiased evaluation of the code.

Initially, we adopted concatenation to generate the audio, and the text processing was done on a granular level to account for the complexity of language/text in email content. After discussing with the client, the audio quality produced by this method was decided to be unsatisfactory, so we searched for alternative methods. We ultimately decided on the gTTS library for TTS. This library produces higher quality speech than our previous method. However, in isolation, the gTTS library could not address all of our text-preprocessing and speech synthesis needs (Sections 5.3.1 and 5.3.2). Therefore, our final strategy was to combine the gTTS library with our previous text-preprocessing code as well as some speech post-processing on the phrase-level.

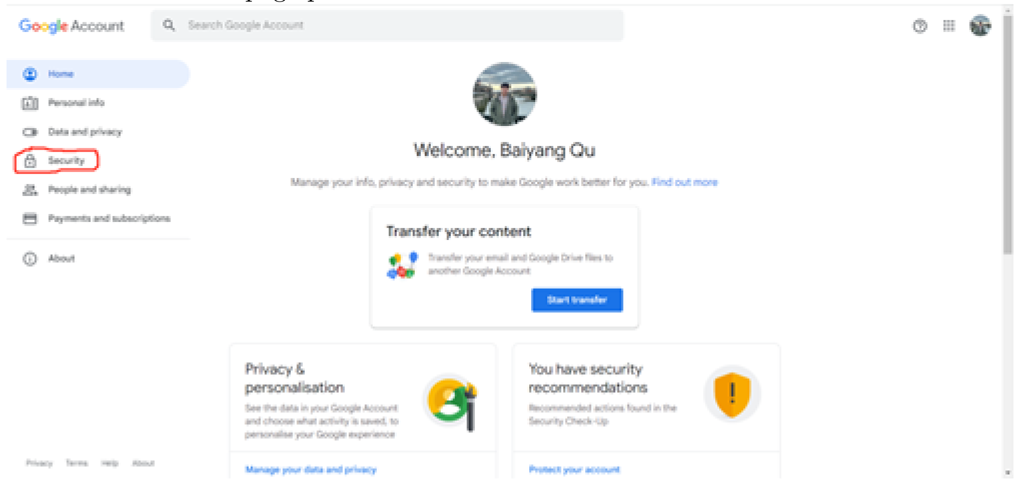
Overall, we learned how to collaborate, and manage and plan projects. We improved our technical skills and learned how to problem solve. We learned the importance of planning, and that lack of planning in one aspect will lead to later problems. For example, we began the testing process later than was ideal.

13 Appendix

13.1 User guide

Subsequent log-ins (after set-up):

1. Open the localhost:8083/audioEmail/login/loading/
2. You will see the webpage pictured below



Welcome, Baiyang Qu

Manage your info, privacy and security to make Google work better for you. Find out more

Transfer your email and Google Drive files to another Google Account

See the data in your Google Account and choose what activity is saved, to personalise your Google experience

Recommended actions found in the Security Check-Up

3. Input email address, e.g. ‘crowds723@gmail.com’
4. Input password, e.g. ‘omitted’
5. Choose account type, between ‘personal’ and ‘work’

Log out and log in with another account:

1. Click ’log out’ on the left hand side, you will be taken to the login page shown above
2. Input email address and password
3. Choose account type

Send Mail between work and personal account (two accounts in two browsers):

1. Open two web pages in order to log in to your accounts
2. In one account click ’Write Message’ on the left hand side, write the recipient, subject and body, and then send
3. In the other account, click Sync, or logout and login again - you should see the email from the other account in your inbox

Reading emails, or having emails read to you:

1. Navigate to your inbox, you should see a list of emails
2. Under the ’Operation’ column, you should see two buttons for each email: ’Read’ and ’Detail’
3. To view email text, click ’Detail’, this will take you to a different page that contains the email content as text
4. To have the email read to you, click ’Read’, this will open a playable window on the inbox page. Clicking the play button (the arrow icon) should begin playback, and you should hear speech
5. Volume can be adjusted by clicking the loudspeaker icon and dragging the slider up or down

Change settings:

1. Navigate to the settings menu, on the left hand side
2. From here you can change username, email address, voice type (accent), and you can set a scheduling time
3. After adjusting settings, click 'Save' to keep your changes, or 'Reset' to return to the default settings

Set scheduling time:

1. Follow step 1 in 'Change settings'
2. In the drop down menu titled 'Schedule Time', choose the time that you want the application to start reading emails
3. Save and exit the settings page
4. To hear playback at your selected time, you will need to have logged into your account by the selected time
5. (Please note that selecting all unread emails to be read uses similar functionality, however, playback may take longer to begin)

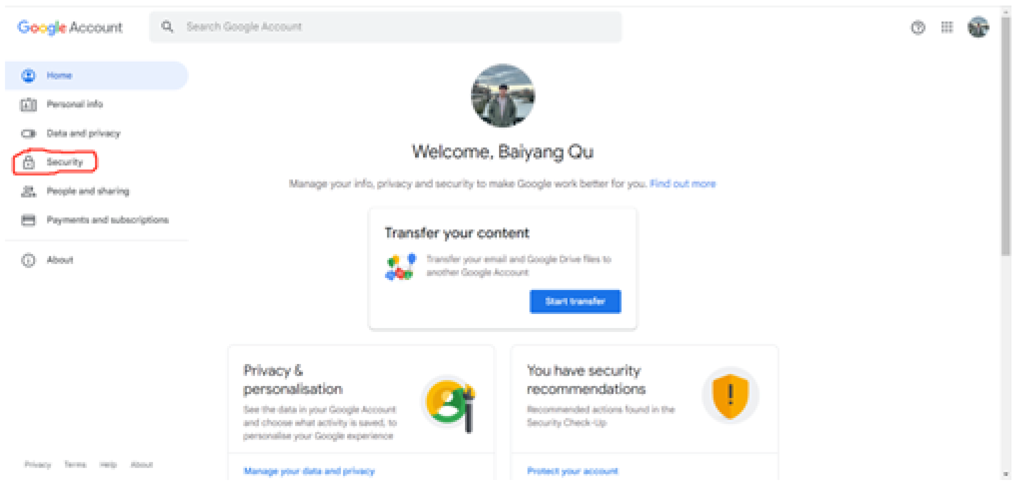
13.2 Setup Guide

This guide applThe script 'dependencyHandler.py' must be executed by the user. It checks the system for the necessary imports, and then installs any missing dependencies.

1. How to get the app password

   First time login: The user will need to get an 'App password' from Google. Instructions may differ slightly if the user does not have a Google account. However, the user can search for 'App Password' from the homepage of their email provider.

   - Navigate to Google account. The web address should start with 'https://myaccount.google.com/'.
   - If not enabled, enable 2-step Verification, which can be found on the Security page under the heading 'How you sign in to Google'.
     - Google Account → Manage Accounts



Welcome, Baiyang Qu

Manage your info, privacy and security to make Google work better for you. Find out more

Transfer your content

Transfer your email and Google Drive files to another Google Account

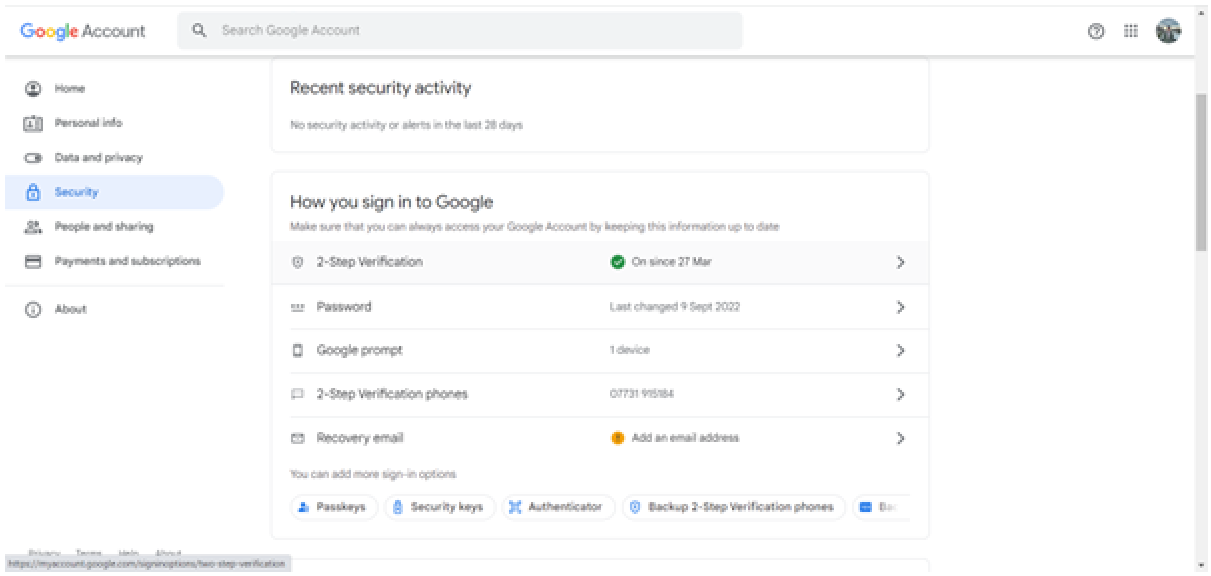
Privacy & personalisation

See the data in your Google Account and choose what activity is saved, to personalise your Google experience

You have security recommendations

Recommended actions found in the Security Check-Up

   - Once you click 'TURN ON' on the 2-step Verification, you will be prompted to enter your mobile phone number, with the choice of receiving either a voice or text message.
   - Follow the instructions and enter the verification code by whatever means you selected in the previous step.
   - Navigate back to the Google account homepage.
   - Search for 'App Passwords', and a suggestion with the same name should appear under 'Security'. Click this suggestion.



Recent security activity

No security activity or alerts in the last 28 days

How you sign in to Google

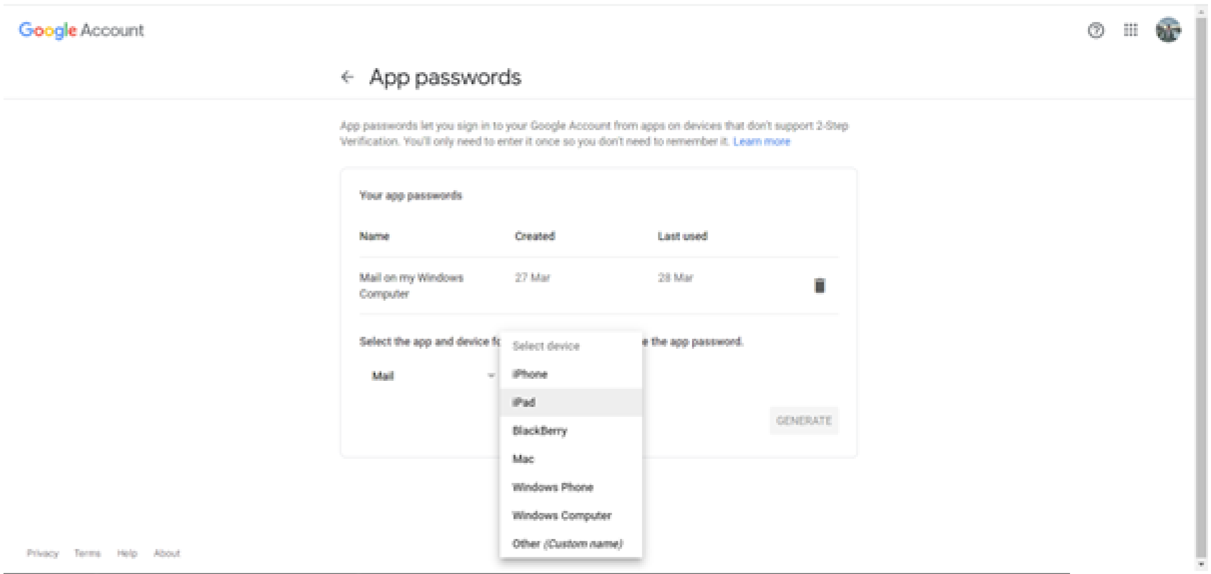
Make sure that you can always access your Google Account by keeping this information up to date

2-Step Verification	On since 27 Mar
Password	Last changed 9 Sept 2022
Google prompt	1 device
2-Step Verification phones	07731 915184
Recovery email	Add an email address

You can add more sign-in options

https://myaccount.google.com/signinoptions/two-step-verification

- You will see two drop-down menus called ‘Select app’ and ‘Select device’. Under ‘Select app’, you must select ‘mail’. Under ‘Select device’ you should select the device on which you wish to use MyAudioEmailr. For example, if you want to open MyAudioEmailr on an iPhone, then you should select ‘iPhone’.
- Once you click ‘GENERATE’, you will be given an app password. You can now continue following the steps below.



Google Account

App passwords

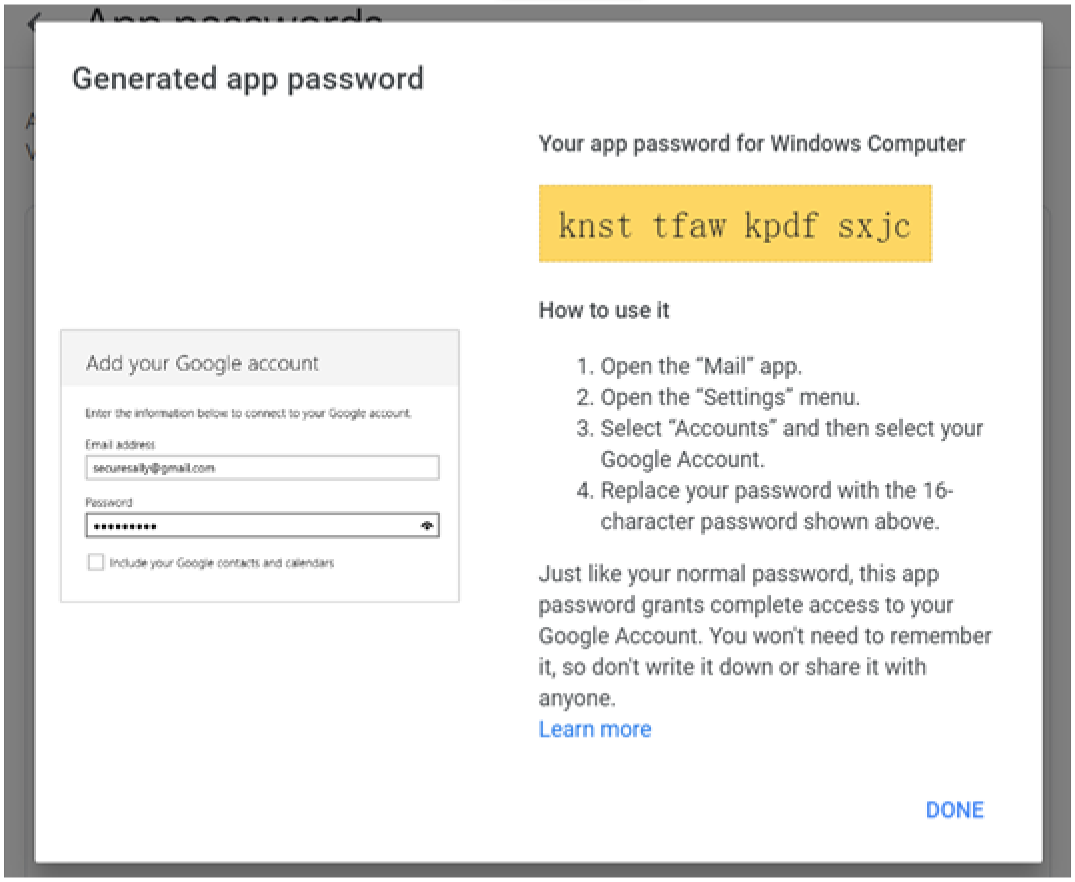
App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. Learn more

Your app passwords

Name	Created	Last used
Mail on my Windows Computer	27 Mar	28 Mar

Select the app and device f… e the app password.

Mail



Generated app password

Your app password for Windows Computer

knst tfaw kpdf sxjc

How to use it

1. Open the “Mail” app.
2. Open the “Settings” menu.
3. Select “Accounts” and then select your Google Account.
4. Replace your password with the 16-character password shown above.

Just like your normal password, this app password grants complete access to your Google Account. You won't need to remember it, so don't write it down or share it with anyone.
Learn more

Add your Google account

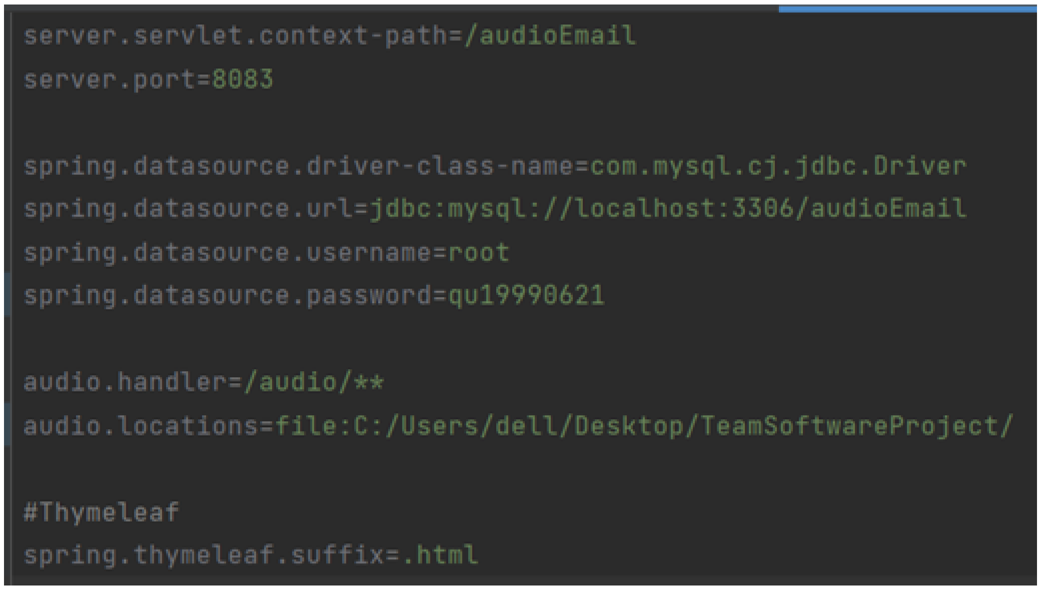
Enter the information below to connect to your Google account.

Email address: securesally@gmail.com

Password: ••••••••••

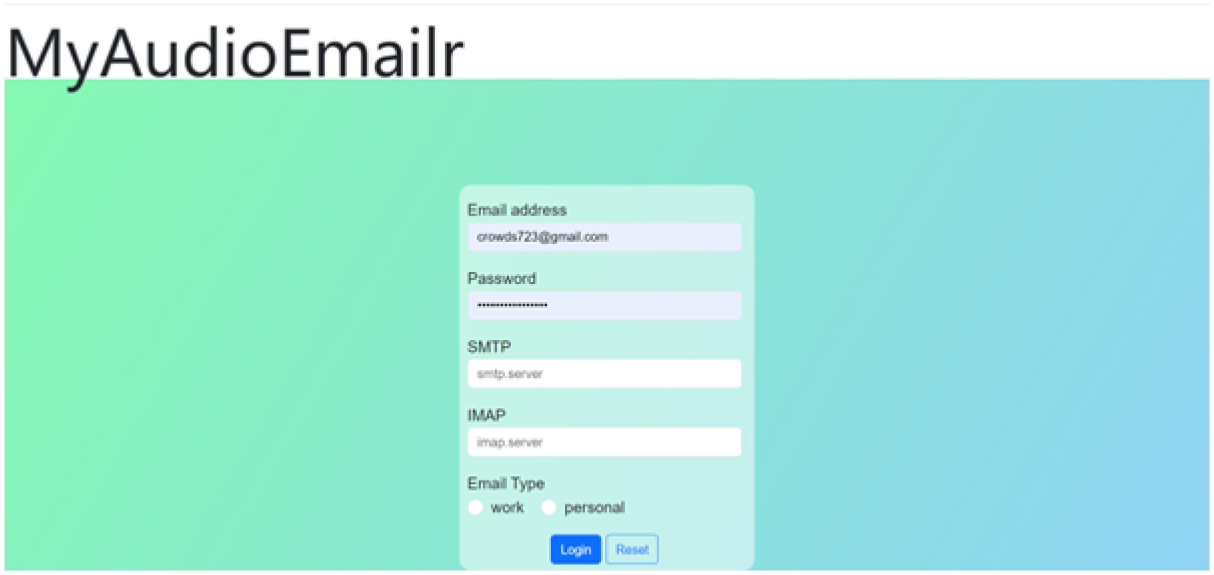
2. Python files for deployment on the server
3. Pip install packages

4. Run python script
5. Configuring the database
6. You need to configure the application.properties file, which including the file path and the database address

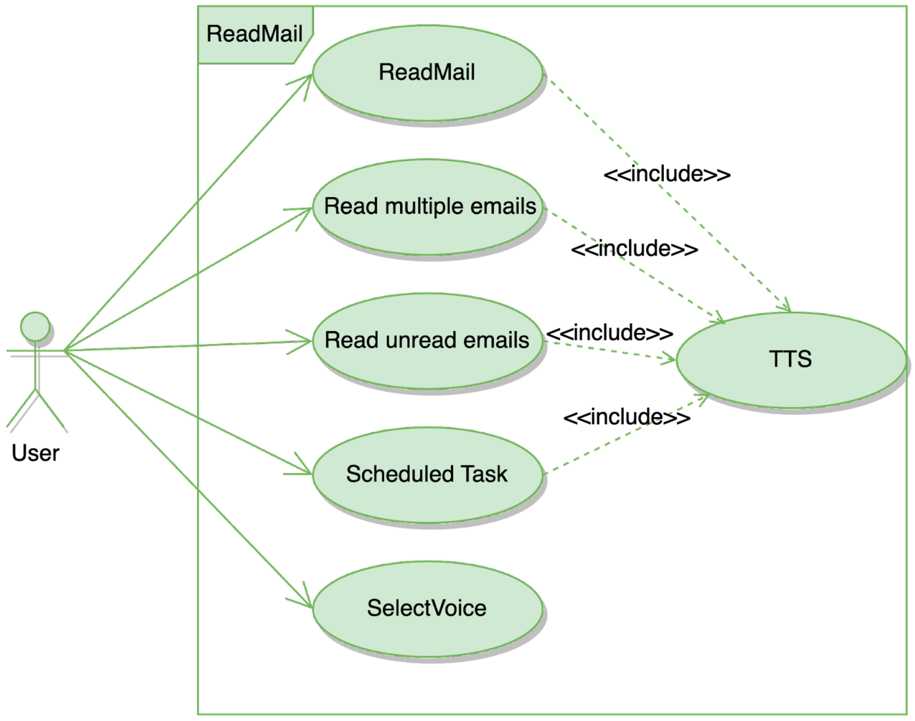


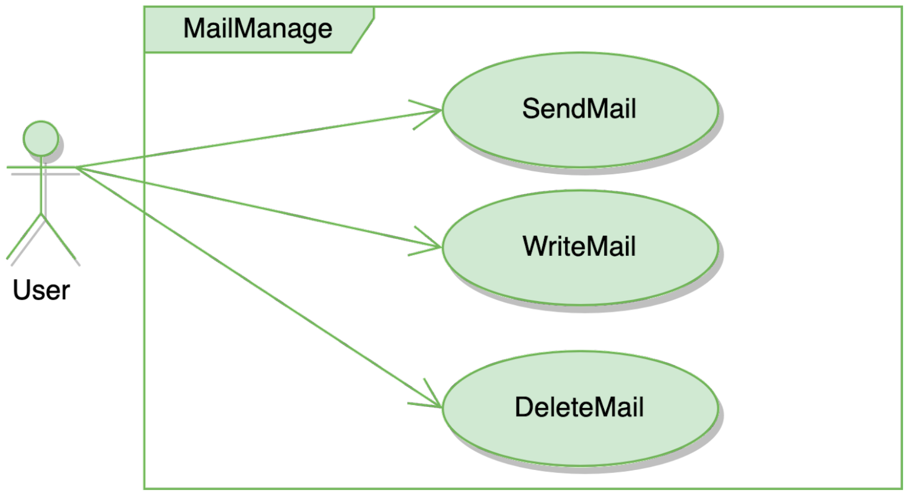
```
server.servlet.context-path=/audioEmail
server.port=8083

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/audioEmail
spring.datasource.username=root
spring.datasource.password=qu19990621

audio.handler=/audio/**
audio.locations=file:C:/Users/dell/Desktop/TeamSoftwareProject/

#Thymeleaf
spring.thymeleaf.suffix=.html
```

7. Build the project into a jar package
8. Deploy the jar package on the server (linux) and use the bash script to start the jar package.
9. Or run the email part via Intellij directly(without building and deploying)
10. Open a browser to access the project address.



13.3 Class Diagram





13.4 Minutes for week3

Meeting on Thursday 23/02/2023 at 11:30 And ended around 12:45
This week:

- Mingqing is working on git lab and writing documentation about the software we will write. He will share it with the group later.
- Ziyu is designing the database structure (MySQL). Ziyu and Mingqing are planning to work on the back end.
- Baiyang is supporting Ziyu and is learning MySQL.

- Rachel will go through the agenda - she has watched the pre-recorded content and made notes on it (shared in WhatsApp group). Rachel has suggested a collaborative agenda and she would like to be in charge of editing the agenda.
- Nana is working on the spec end with Mingqing and Ziyu.
- Georgia is researching TTS..

General points:

- Division: 3-3 split - Nana, Mingqing, Ziyu on more technical parts (database and Java). Baiyang, Rachel and Georgia on TTS.
- We will store all documentation on Google Documents.
- Everyone should have their own branch on Git Lab. Use of Git Lab off-campus requires a VPN.
- Okay to use Word, but it is common practice to use markdown or LaTeX.

Next week:

- Mingqing and Nana are going to work on the sending and receiving email functionality.
- Ziyu will work on building the environment
- Baiyang, Rachel and Georgia will work on the TTS function. It will be written in Python. We need data: diphones, any open-source set, prosody and word sense disambiguation possibly from NLTK. Baiyang will work on the schematic while Georgia and Rachel will work on gathering data.
- On Tuesday we will talk about the key features. We will clarify with Ola regarding preferred language.

13.5 Additional test cases

13.5.1 Test Case 5: Unread email state

Test Case 5: Unread email state

- Test step:
  ASend a new email
- Expected result:
  The email should be marked as unread before opening or reading it
- Actual result:
  The email is marked as read once user receives it
- Commit record:
  1568f9379e775d4ce0e79ae1d60613de43993be6

13.5.2 Test Case 6: send email

Test Case 6: send email

- Test step:
  Try to send an email via the ‘write message’ button
- Expected result:
  Receive the email in the account specified in ‘recipients’
- Actual result:
  It shows ‘send failure’ first, but I can get the email sent to my personal account
- Commit record:
  061c97a4000c19264807d8a924efc49b7fa5b4d0

13.5.3 Test Case 7: testChangeStatus

TestCase 7: testChangeStatus

- Test intput:

```
Test Input:
String name = "mq";
String senderAddress = "crowds723@gmail.com";
String receiverAddress = "mzhang122@sheffield.ac.uk";
String smtp = "smtp.gmail.com";
String imap = "imap.gmail.com";
String deviceId = "202303281710-diive";
```

- Expected result:
  The status of all emails in this account should be changed
- Actual result:
  Test case pass

13.5.4 Test Case 8: testInitEmails

Test Case 8: testInitEmails

- Test intput:

```
Test Input:
String name = "mq";
String senderAddress = "crowds723@gmail.com";
String receiverAddress = "mzhang122@sheffield.ac.uk";
String smtp = "smtp.gmail.com";
String imap = "imap.gmail.com";
String deviceId = "202303281710-diive";
```

- Expected result:
  The user's information is inserted into the system_account table, and the last ten emails received are inserted into the mail_info table
- Actual result:
  Test case pass

13.5.5 Test Case 9: syncTest

TestCase 9: syncTest

- Test steps:
  Send an email to the currently logged in user
  Wait 10 mins then check if there is a new mail in the database
- Test intput:

```
Test Input:
String name = "mq";
String senderAddress = "crowds723@gmail.com";
String receiverAddress = "mzhang122@sheffield.ac.uk";
String smtp = "smtp.gmail.com";
String imap = "imap.gmail.com";
String deviceId = "202303281710-diive";
```

- Expected result:
  The user's information is inserted into the system_account table, and the last ten emails received are inserted into the mail_info table
- Actual result:
  Several emails appear, but not the latest from the database
- Commit record:
  54619c53bb993dcb22822202dd3d5bcfb9a235c7

13.5.6 Test Case 10: playUnreadMail

Test Case 10: playUnreadMail

- Test step:
  After login click the playUnreadMails button
- Expected result:
  Users do not need to wait for all audio transcriptions to be completed
  All unread mails can be read one by one
- Actual result:
  Users need to wait for all emails to finish processing before any are read
- Commit record:
  0a1e8daffc97e55efb02339785b5393123ecff28 0295890ef4a22bcc470cc68acf12439a300b7e8f

13.5.7 Test Case 11: playSpecificMail

Test Case 11: playSpecificMail

- Test step:
  After login click the playSpecificMails button
- Expected result:
  Users do not need to wait for all audio transcriptions to be completed
  All unread mails can be read one by one
- Actual result:
  Users need to wait for all emails to finish processing before any are read
- Commit record:
  0a1e8daffc97e55efb02339785b5393123ecff28 0295890ef4a22bcc470cc68acf12439a300b7e8f

Test Case 13: Increased volume from exclamation marks Test step:
Expected result:
Actual result:

13.5.8 Test Case 12: Handling punctuation

Test Case 12: Handling punctuation

- Test step:
  Passed an html string containing punctuation not handled by text_preprocessing.py (e.g. { or), with some punctuation in a sequence (e.g. ???), to tts using the test.py module
- Expected result:
  These punctuation marks should be pronounced (e.g. "curly bracket") or omitted
- Actual result:
  A given punctuation mark is passed to the gTTS class as a phrase, raising an AssertionError, due to there being nothing pronounceable
- Commit record:
  f9240803a42d9fd44e4615265be527d7ebc3f4fb eb4170f9bd747d788f9a44bce71b10e60bf15cfd

13.5.9 Test Case 13: Increased volume from exclamation marks

Test Case 13: Increased volume from exclamation marks

- Test step:
  Using test.py, I passed a string containing some exclamation marks to tts
- Expected result:
  These punctuation marks should be pronounced (e.g. "curly bracket") or omitted
- Actual result:
  Test case pass

13.6 References

- Beautiful Soup 4.12.2 [software]. Leonard Richardson. (2007)
  Accessed 10 May 2023
- gTTS 2.3.2 [software]. Pierre-Nick Durette. (2020)
  Accessed 4 May 2023
- Meyer, A. N., Barton, L. E., Murphy, G. C., Zimmermann, T., & Fritz, T. (2017). The work life of developers: Activities, switches and perceived productivity. IEEE Transactions on Software Engineering, 43(12), 1178-1193
  Accessed 10 May 2023
- NLTK 3.8.1 [software]. (2023)
  Accessed 4 May 2023
- Pycharm Professional 2022.3.2 [software]. JetBrains. (2022)
  Accessed 10 May 2023
- Soundfile 0.12.1 [software]. Bastian Bechtold. (2020)
  Accessed 10 May 2023