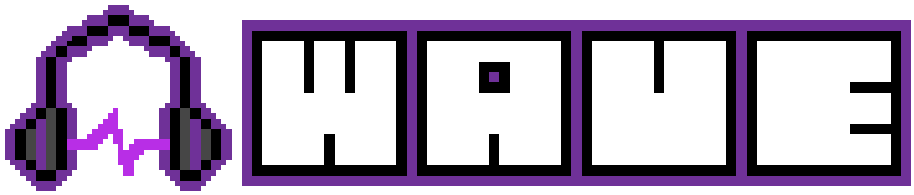


INF-2900 Exam Project

Members:

Tarek Lein
Iver Mortensen
Marcus Ryan
Amund Strøm
Theodor Tredal

GitHub Classroom: Team 14

INF-2900: Software Engineering

Faculty of Science and Technology

August 22, 2024

Contents

1 Introduction

The report below shows the development of our web-based music application that will offer the users an opportunity not only to listen to other people's tracks but also to upload their own tunes. It will enable users to mix their personal creations with established music, so they can have a unique and personalized listening experience.
We used Scrum within our agile development to ensure a smooth and productive development processes. It encourages incremental and iterative development, by allowing constant feedback and adaptation. Our well-thought-out process mainly comprised sprint planning, reviews, and retrospective meetings that created effective collaboration and communication within the team. This allowed all the members to be on the same page with the goals of the project and pull their contributions off efficiently.
Additionally, the report will cover the lessons learned, challenges, and future work.

---

2 The Project and the Vision

In our first meeting we discussed different possible ideas for what the product of our project could be 10.6. We landed on the idea of a web-based music application that would allow users to upload their own, or known songs. We then created 5 personas that would represent potential users 10.2. From the personas we got an image of what features would be good to implement to satisfy their needs. The 6 features we established was song search, playlist creation, uploading of mp3 files, listening history, shuffle and queue, and the functionality to see other users' playlists.

2.1 Product Vision

For our product vision we follow the template that comes from the book Crossing the Chasm by Geoffrey Moore where he suggests a structured approach based on these keywords:

- FOR (target customers)
- WHO (statement of the need or opportunity)
- THE (PRODUCT NAME) is a (product category)
- THAT (key benefit, compelling reason to buy)
- UNLIKE (primary competitive alternative)
- OUR PRODUCT (statement of primary differentiation)

With this structure we aim to answer the three fundamental questions that the product vision should answer: Sommerville (2021).

1. *What* is the product that you propose to develop? What makes this product different from competing products?
2. *Who* are the target users and customers for the product?
3. *Why* should customers buy this product?

FOR people who listen to and/or creates music, WHO needs an application to upload their own or known songs. THE WAVE web application is a platform THAT provides a place for music listeners to create tailored playlists containing their favorite tracks, alongside their own. UNLIKE other music streaming software products, OUR product allow users to blend their own compositions with established tunes, without the need of being a recognized artist.

Its clear that the product vision answers the three fundamental questions listed above:

1. *What* A web application provides a place for music listeners to create tailored playlist containing their favorite tracks.
2. *Who* The product is aimed at people who listens to and/or creates music, and needs an application to upload their own or known songs.
3. *Why* The product functionality that sets us apart from other applications is the ability for users to upload their own songs without being a recognized artist,

2.2 Goals

Based on everything written above it is clear that our main goal was to create a web-based music application. We then split this goal into sub-goals that we want our product to achieve. These are goals for the development process and goals for the finished product.

2.2.1 Goals for Development Process

We conducted a set of goals we wanted to achieve by using agile development with scrum for the development process.

Iterative Development: As scrum employs Iterative Development where projects are broken down into smaller increments called sprint, we ensure that we'll deliver working software at the end of each sprint. This will allow for continuous feedback and adaption.

Collaborative Teamwork: Scrum emphasizes collaborative teamwork between each member of the team with different skills and perspectives. By ensuring a collaborative environment, we'll get collective knowledge that can be used to further enhance the product.

Adaptability to Change: Scrum prioritizes responding to changing requirements and priorities throughout the development process. This is embedded in Scrum's iterative nature. As a team we want to be adaptable to changes that naturally occur during development.

Continuous Improvement: A core principle of Scrum, where the importance of reflecting on the development process and making incremental changes to optimize the development. By striving for continuous improvement, we can adapt and improve our practices to deliver better results with each iteration

2.2.2 Product Goals

Features: We want our finished product to include all the 6 features we established based on the personas 10.2 at the start of development. We will allow alterations and possible improvements of these features.

Security and HTTPS: The security of the product had high priority and was a crucial goal at the start. The user should be able to securely use our application without the need to worry about their credential being exposed. We established the goal of connecting the server with HTTPS to ensure the ability for users to securely use the application from their home computer.

User Friendly: We wanted our end product to be user friendly to ensure a pleasant user experience. This would include an aesthetic design, simplicity, and ease of use. The application should also be interactive, by telling the user if an error have occured, for instance if the back-end fails while uploading a song.

Future goals: The future goals refer to the objectives we established at the start of development, which we anticipate will be achieved as we continue working on through the next iterations. This includes expanding the 6 features, the addition of new features, improving and optimizing the application based on user feedback.

3 The Development Process

3.1 Agile Development

Agile methods such as Extreme Programming (XP) and Scrum were utilized in our project to carry out and manage the development process. These methodologies focuses on things like step by step development, adaptability and continuous improvement which aligns well with the requirements of our project.

3.1.1 Scrum

We followed some of the scrum methodologies like sprint planning, sprint review, sprint retrospective meetings, backlogs and incremental development. We also had a scrum master which we changed each sprint.

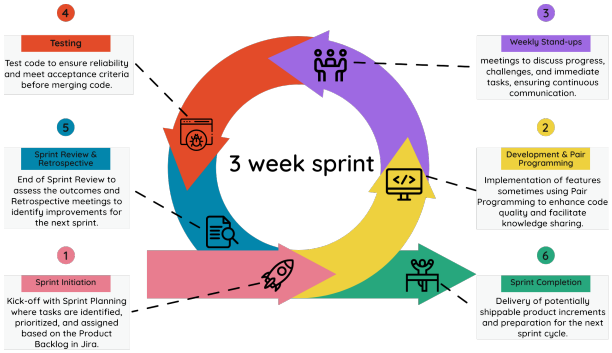


Figure 1: Agile Development Cycle

3.1.2 Extreme Programming (XP)

During the first sprint, we used Pair Programming to get a grasp of the main frameworks we were going to be using to build our application. This approach not only improved our code's reliability but also helped with knowledge sharing between team members which was important for maintaining consistency when we were developing.

3.1.3 User Stories and Jira Integration

We used the online scrum tool Jira as our project management tool, there we organized our work into Epics ->User Stories ->Tasks. This helped us segment large goals into smaller manageable tasks, making it easier to track progress and prioritize work effectively. The user stories (10.1) were important in keeping the team aligned with the end-users needs and expectations.

3.1.4 Test-Driven Development (TDD)

We tried to adopt Test-Driven Development to make sure that all new features were reliable and passed all test before they were added to the main branch on GitHub.

But due to our inexperience in developing an API service, it was difficult to know how the output was going to be ahead of time, which is why we decided to not continue using TDD.

3.2 Backlog Planning, Sprints, and Retrospective Meetings

When managing our Agile project we followed a clear approach for planning the backlog, executing sprints, and holding retrospective meetings. This section describes the methods and tools that were key to running and continuously improving our project.

3.2.1 Backlog Planning

The product backlog is a list with tasks that contains things such as bugs, feature requests and improvements. The backlog is prone to change in the way that we add more tasks to it since we are going to find bugs and potentially new features that were not considered at the beginning. These tasks are prioritized based on how important they are to our application. Using this structure we were able to stay within the projects scope and requirements.

At the end of each sprint we collectively decided on what features were important and should be implemented next, based on this we could figure out which tasks needed to be done and inserted them into our next sprint iteration.

By utilizing the story point system, we could approximate and distribute the backlog tasks evenly among the team members, this in turn helped balancing the workload throughout the sprints.

3.2.2 Story Points

When assigning story points to backlog items, we initially used a scale between 1 and 6 points, where 1 point equaled a day's work, and 6 points equaled six days work. With this scale we gave all items story points based on the predefined time estimates. Based on these time estimations, we assigned each sprint a total of 50 story points. However, during the development process, we noticed that in some sprints, all backlog items were completed ahead of the deadline. As a result, we added more backlog items to those sprints, which is the reason for sprints having approximately 60 points.

3.2.3 Sprints

Our project was split into four main sprints where the first lasted two weeks because it was used as a foundation sprint for pair programming to collectively learn Django and react, and the rest of the sprints were three weeks. Each sprint we assigned a new Scrum Master so that everyone could try out this role and learn about it, this promoted leadership diversity and helped with team involvement. Daily scrum meetings were nearly impossible for us as students, since we have different classes to attend and work, so instead we had weekly meetings every Tuesday and used Discord if we wanted to reach someone outside of the set meeting times, random informal meetings also occurred when we meet each other at school. Tasks for each sprint were strategically selected from our backlog at the end of each sprint, with important functionalities like the back-end API being prioritized in the early sprints. We also had sprint reviews where we presented all the features completed.

3.2.4 Product Backlog and Sprint Cycles

Our product backlog in Jira contained all planned features and tasks, prioritized according to the project's end goal. Sprint planning sessions were held at the end of each sprint, where the team collectively decided the tasks to be tackled in the next sprint based on the backlog and current project status.

Sprint	Amund	Iver Mortensen	Marcus Ryan	Tarek Lein	Theodor Tredal
2	13.0	13.0	12.0	13.0	15.0
3	12.0	13.0	12.0	13.0	14.0
4	13.0	12.0	14.0	15.0	12.0

Table 1: Story points per member per sprint (not including sprint 1 because it was used as a foundation sprint)

3.2.5 Retrospective Meetings

At the end of each sprint we had retrospective meetings to review our work and look for ways to improve. These meetings were important for continuous improvement, helping us to quickly refine our methods and practices. During these meetings we asked our selves some of the typical retrospective meetings questions like "What went well" and "What could be improved". The insights from these sessions led to direct improvements in how we worked together and the outcomes of our project.

This figure presents the "Song Search Feature" user story, categorized under the epic "User Features & Interaction" (ID no. 121). All Jira related pictures are included in the Appendix 10.5.1

The user story's description details the objectives and requirements of the feature. We can also see the associated sub tasks which represent individual tasks required to complete the user story.

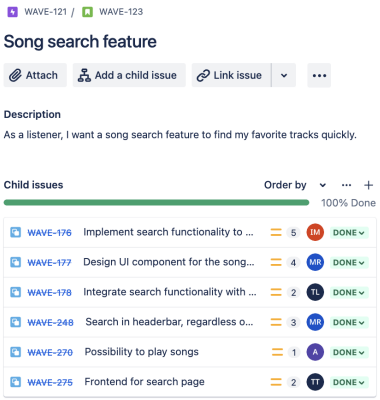


Figure 3: A user story on Jira

3.2.6 Visual Aids

Below is a Gantt chart and a picture of a user story from Jira to provide visual representation of our projects timeline.

Figure 2: Gantt Chart illustrating the project timeline and key milestones across the four sprints.

3.3 GitHub Workflow

We used GitHub as our tool for version control and collaboration during development. In our directory we had the front-end and back-end separated into two folders. Within the front-end we modularized the code, by separating each web-page into folders containing their respective files. Functions used in multiple places or prone to change was implemented inside a separate components folder. This structure reduced the amount of code within each file, which was done to mitigate the number of merge conflicts, a common occurrence when multiple developers work on the same project.

We had a development branch for each member. We would then select a task each from the backlog and implement it on our own branch. Once a feature was completed and thoroughly tested, we then merged our independent branch into the main branch. This made conflicts less prone, and ensured a stable main branch after a successful merge.

Despite of the enforced measures, merge conflicts still arose. We addressed these during informal meetings and code review sessions. This did not only resolved the conflicts, but also enhanced the teams overall understanding of the project as new features where added.

3.4 Testing

We used a python library called Coverage to see how much of our code was covered by tests. Coverage generates a HTML file that shows you exactly what code were ran by the tests, by using this visualization we went from 60% coverage to 97% as seen in figure table below which was a substantial improvement. 2

Test-Driven Development was initially tried in the beginning as mentioned in the Agile development section 3.1.4. The test-driven approach did not fit our style of development, as writing tests before the code would prove to be too time consuming, as the technicality of the features were ever-changing. TDD would thus take time away spent on actually developing the features. As the code were ever-changing, the tests would therefore become futile, and would have to be rewritten. after the second sprint retrospective meeting, the TDD approach were dropped, in favor of testing the code when implemented.

Later on in our development process we found out about continuous testing with the help of GitHub Actions, and wanted to implement it. This significantly improved our development efficiency, because it reduced the need for manual testing since it automatically ran tests on GitHub after each push. In addition it wouldn't allow a push/merge to be committed if the tests failed, meaning that the main branch would always be stable and deployable.

The following table represents sprint four's coverage report generated by Coverage.py, indicating a high level of code execution verification across various modules of our back-end in both the API and authorization components.

Table 2: Code Coverage Report

Name	**Stmts**	**Miss**	**Cover (%)**
user_Api/models.py	21	0	100
user_Api/serializers.py	11	0	100
user_Api/tests.py	208	0	100
user_Api/views.py	162	7	96
user_Authentication/models.py	20	3	85
user_Authentication/serializers.py	24	0	100
user_Authentication/tests.py	112	0	100
user_Authentication/validations.py	21	3	86
user_Authentication/views.py	80	6	92
TOTAL	659	19	**97**

As shown in the figure almost all our components achieved a perfect score of a 100%, with an overall coverage of 97%. The few components that did not reach this were because of edge cases or exceptional conditions where tests would be redundant.

In the end we ended up with 49 tests in total between our song management and authorization API.

---

4 Design

4.1 Framework and Languages

React was used to create the front-end of the application. React is a JavaScript library which uses components that enabled us to implement a dynamic web-application.
For styling we used SCSS, an extension of CSS that allow the use of global variables and nesting, which reduces the amount of redundant code. This enabled us to work faster while maintaining a more consistent UI. Bootstrap and Chakra UI was used in the designing process of the front-end. The large library of icons in Bootstrap and the ready to use components in Chakra UI made the creation of aesthetic designs more efficient.

For the back-end, we chose Django together with the REST Framework. Django is a Python web framework which allow for efficient back-end development. It offers robust and secure back-end functionalities and comprehensive middleware support which is essential for proper handling of the data interactions between the front-end and the database.
Django's REST Framework is used to build these Web API interactions for our React front-end. It supports RESTful request handling, providing tools to create endpoints for HTTP methods, which are essential for CRUD operations (Create, Read, Update, Delete) related to user and music data management.

4.2 Design Aesthetic

The design aesthetic refers to an application's visual style and pleasing qualities. It involve elements like color, fonts, images, layouts, and subtle animations that all combines to represent the overall look and feel of the application. We can increase the user experience and brand identity by prioritizing the websites' aesthetics, making it more intuitive, engaging and it might even impact users loyalty and satisfaction.

4.2.1 Brand identity

While developing the aesthetic and identity for WAVE, we wanted to stand out from competitors in the market. This led us to choose a purple color palette, which sets us apart visually from other platforms like Spotify, Tidal, and Soundcloud, each with their own personality and color scheme. You'll find the purple theme throughout the application, from buttons and text to containers and backgrounds. To keep the overall interface clean and avoid visual pollution, we used gray and black shades for most of the application, while purple remains our signature color.

When creating our logo, the goal was to show our place in the market clearly as a music platform. Therefore the logo includes a headset, representing music, and the name "WAVE", symbolizing sound waves. The logo is also framed by a purple outline to strengthen our brand identity, as shown in the images below.

(a) The WAVE logo.

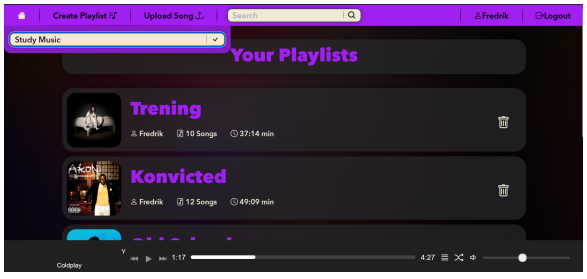
(b) The WAVE icon.

(c) The WAVE text.

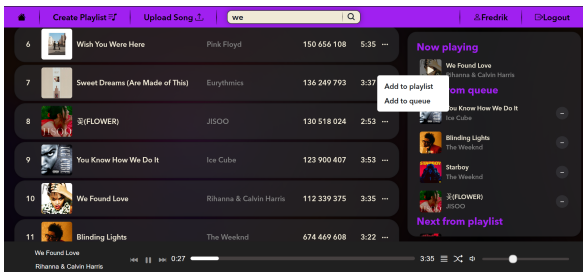
Figure 4: The WAVE logo and it's respective components.

4.2.2 Design Elements

When designing the elements in our application we focused on building a clean and professional website, and maintaining visual consistency in the elements across the pages. Being visually consistent is crucial for ensuring an intuitive and seamless user experience, while further increasing brand identity as the style in our application becomes unique to our brand. This is why you'll see the consistent use of containers, background, and fonts illustrated in the figure 5 below.



(a) The homepage.



(b) The search page.

Figure 5: The web application being visually consistent on two different pages.

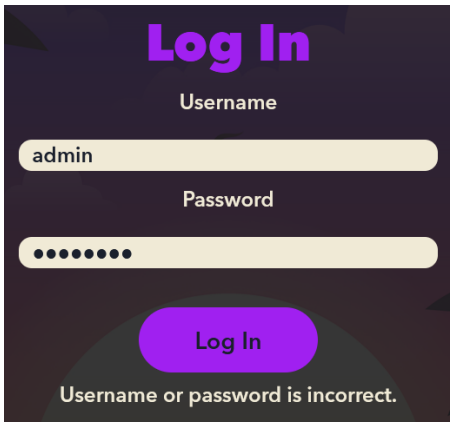
As seen in the images fonts are consistent and have visible colors contributing to readability, containers have rounded corners with dark shading making the application predictable, and intractable components are clearly outlined and provides users with predictable and reliable feedback. This visual consistency combines to make a user-friendly experience when navigating the application, since the interface maintains an cohesive design from page to page.

4.2.3 Responsive design

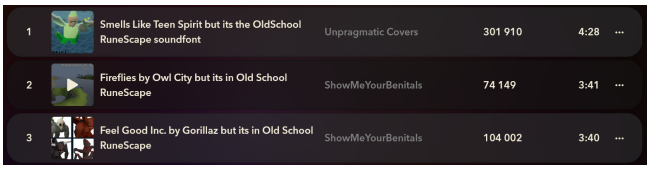
A responsive design ensures that an application adapts seamlessly across various screen sizes. To implement this we adopted good practice approaches such as using relative units like percentages for layout dimensions, which allow containers, images, and text to adjust dynamically. Correct use of the CSS "position" property is also important, it's generally recommended to avoid using "absolute" and "fixed" positions as they can cause layout issues. Following such guidelines, our application functions well across different desktop screen sizes.

4.2.4 User Feedback Mechanisms

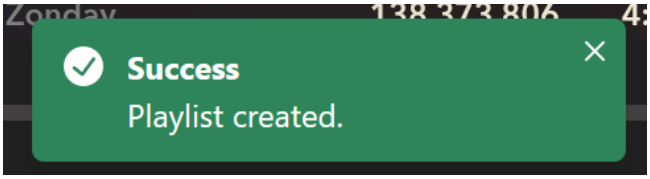
Implementing clear feedback mechanics ensures that the user feels in control and understands the outcomes of their interactions within the application. These mechanisms include visual cues such as hoverable components changing colors and signaling clickable objects, informational cues prompting the user if they enter their password incorrectly, and cues utilizing pop-up modules that inform the user whether an action was successful or not, such as when uploading a song. These feedback mechanics create an responsive interface that enhances the overall user experience.



(a) Incorrect username or password.



(b) Hover-effect for song.



(c) Successfully creating a playlist.

Figure 6: Feedback mechanics in our application.

4.3 Front-end Structure

The front-end structure controls how the user can interact with the application and plays a crucial role in defining the user experience. It involves the layout, design, and behavior of the user interface, directly impacting how users perceive and engage with the application.

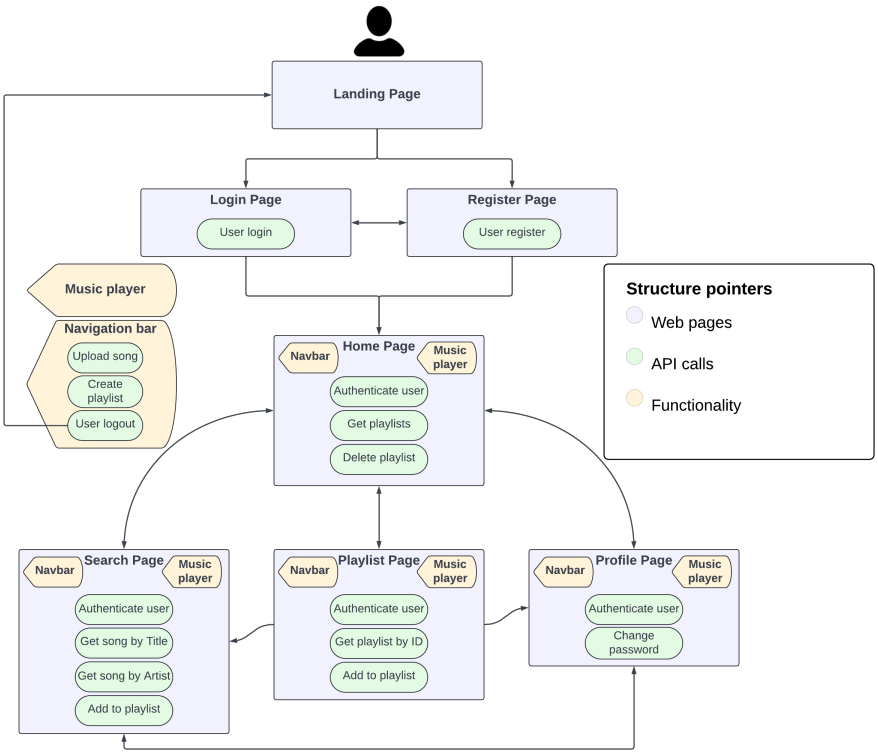


Figure 7: Front-end Architecture of the Wave Music App.

As seen in figure 7 it illustrates the user journey and presents the available API calls on each page, functioning as the features. It breaks down the application into separate pages and components and provides a map of how the user can navigate between the different pages.

Navigation bar & **Music player**: These are common elements across multiple pages that requires authentication and introduces specific functionalities. The navigation bar allow users to navigate to other web pages, upload songs, create playlists, and log out. The music player provides controls for playing, pausing, skipping and navigating through songs.

Landing page: The landing page is the entry point of the application containing general information about the features available and enables the user to login or register.

Login page & **Register page**: Users can either log in with an existing account or register a new account, both of which will navigate to the home page .

Home page: Once authenticated, users enter the home page where they can manage their playlists, such as viewing their current playlists and deleting them.

Search page: The search page allow the users to search for new songs, based on song title and artist name, and adding them into playlists.

Playlist page: The playlist page is available through the home page by navigating to a specific playlist, and it displays songs in the current playlist and allow the user to add songs to other playlists.

Profile page: Users can view details about their profile and change their password.

4.4 Back-end Structure

The design of the back-end is made to handle user interactions in a stable and consistent way. Django and the REST framework is used to effectively support user authentication and manage different song and playlist related actions that interact with the SQLite database.

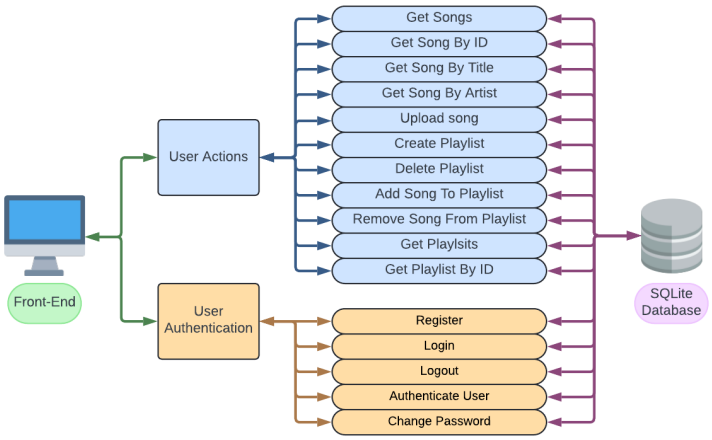


Figure 8: Back-end Architecture of the Wave Music App.

4.4.1 Song Management

The bridge between the user and the songs in the database is provided by the song management system. This system gives the user the abilities to search, add, and manage songs from the front-end.

Get Songs: Retrieves all songs from the database.

Get Song By ID: Retrieve a song based on its ID within the database.

Get Song By Title: Enables users to search for specific songs using their title.

Get Song By Artist: Allow users to search and retrieve songs by the artist name.

Add Song: Users can add new songs to the database by providing a YouTube URL.

4.4.2 Playlist Management

Playlist management is the part of the back-end that enables users to customize their music experience. It consist of different endpoints that allow users to create, modify, and delete playlists.

Create Playlist: Allow users to create new playlists in the database.

Delete Playlist: Provides users the option to delete entire playlists.

Add Song To Playlist: Users can add new songs to a desired playlist.

Remove Song From Playlist: Allow users to manage their playlists by removing unwanted songs.

Get Playlists: Retrieves all playlists created by the signed in user.

Get Playlist By ID: Retrieves a playlist based on its ID.

4.4.3 User Authentication

Critical functionalities like managing user identity and security is handled by the user authentication system in the back-end. This is essential for maintaining secure access and personalization by differentiating user requests. The endpoints provide a variety of operations that together ensure proper user management.

Register: Responsible for creating unique user accounts.

Login: Authenticates user credentials and which grants access to user specific content by generating a session token.

Logout: Allow users to sign out of their session, effectively invalidating the session token to preventing access to user content.

Change Password: Allow users to update their current password.

Authenticate User: Verifies the session token to confirm that the page is accessed by an authenticated user.

4.5 Database Design

We've designed an Entity-Relationship (ER) diagram that shows the database design. It displays the relationships between the data entities in our system, and helps in understanding how we link and manage the data within our application. In this section we will be discussing the schema that is presented in the diagram and how it fulfills the needs for our application.

4.5.1 Overview of the ER Diagram

Here we can see the primary tables in our database; User, Playlist, Song, Playlist Song and the relationships between them.

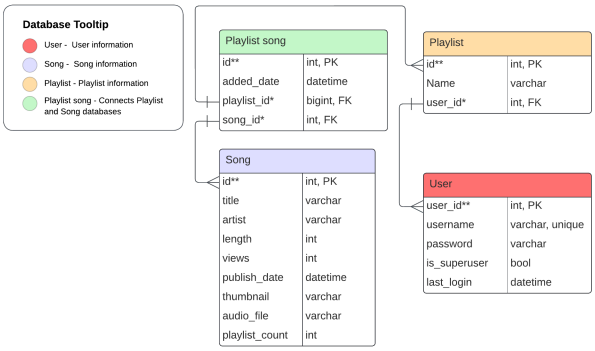


Figure 9: Entity-Relationship Diagram of the Database

4.5.2 Schema Description and Data Support:

Our database schema consists of four main entities, each needed to serve an unique role in our application

User Management: The "User" table handles user data including authentication and profile management.

Playlist Management: The "Playlist" table stores information about user created playlists. It links directly to "User" through a foreign key which is used to determine the ownership of playlists. This relationship allow users to create, modify, and delete their playlists.

Song Management: The "Song" table holds all the data about our available songs like artist, title, length and number of listens. This data allow the users to search for and add songs to a playlist

Playlist-Song Association: The "Playlist song" table is a join table between playlists and songs, representing a many-to-many relationship. This logic allow a single playlist to contain multiple songs and a single song to be a part of multiple playlists.

Data Relationships and Integrity: The schema uses foreign keys to ensure that we have data integrity, it also defines the relationships between tables. For example the foreign key in "Playlist" referencing "User" ensures that each playlist is associated with a registered user. Similarly foreign keys in "Playlist song" link playlists and songs.

4.6 Security Measures

The security architecture of the application is designed to protect both user data and the application integrity. Using Django and its REST framework, the back-end incorporates several robust security measures to ensure secure operations and data handling, which also perpetuates to the security of the front-end.

- **Session Token Authentication:** The security Architecture of the application provides protection to both application and the users data. Using the REST framework provided by Django, created a robust

security that ensures an overall secure operation and data-handling. The security also extends to the frontend

- **Password Hashing:** Password hashing: The users login information are a top priority for our platform. Therefore all user passwords are hashed before stored in the database, this means that our platform never knows the user's login credentials. This will reduce the risk of user's credential's being exposed during a data breach.
- **CORS Configuration:** The application CORS configuration: The application uses CORS headers to manage cross-origin requests. CORS are needed for enabling and restricting resource sharing across different domains securely, utilizing CORS will in turn prevent attacks that exploit browser-side cross-origin interactions.
- **Django REST Framework Security:** Django REST Framework Security: Django's REST framework comes with built-in security features that the application makes use of, including.
  - **Serialization:** We do not have to write our own queries, Django does this for us. This security layer has inbuilt prevention of SQL injections.
  - **Throttling:** Throttling prevents abuse of excessive API requests, by limiting the number of API calls a user can make in a specific time frame.
  - **Permission Classes:** are used to control access to different parts of the API, this is based on the user's authentication status and other custom criteria. This ensures that only users can perform actions they are permitted to.

---

5 Implementation

In the implementation of the application the main functionalities consist of user registration, user authentication and music and playlist management.

User Registration

New users can create a new account by registering with a unique username and a password. This adds the users credentials to the database with the password being hashed.

User Login

When a user is registered they can sign in to their new account. This is done by using their username and password. The users credentials is then crosschecked with the data in the database. If the credentials match, a token is created, signing in the user.

Upload Music

Users can upload new songs by submitting a YouTube URL through the navigation bar. The application uses the Pytube (2024) library to extract MP3 files and relevant song information from YouTube. The MP3 file is used for music playback and the song information is used to display the songs in the front-end.

Search for Music

The user can find new songs by using the search feature available in the navigation bar. This allow for search of the song name or artist. This queries the database, which efficiently finds the desired results.

Music Player

The music player is an interactive component located at the bottom of the page. It offers controls such as play, pause, skip, adjusting the volume, and navigating through the song. The state of the music player is stored in local storage, allowing the music session to persist across the user's active session, thereby improving continuous user engagement.

Create Playlists

The users playlists are displayed on the home page. These can be created using the navigation bar and only requires a name for the playlist, which adds them to the database. Once created, the user can add the songs they desire. If they are done with a playlist, the playlist can be deleted, removing it from the database.

Queue

The queuing system allow users to add songs to a queue. This queue is played before playlist songs. This functionality provides users with greater control over what they listen to next without creating permanent playlists. It's implemented in the front-end and the music player interacts with it to dynamically manage the playback order.

Change Password

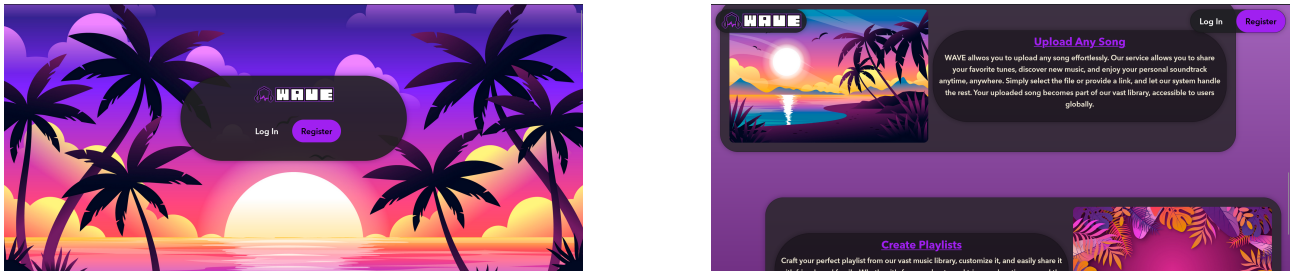
For enhanced security and user experience, the app allow users to change their passwords. Users must provide their current password and a new password, which is then updated securely in the database after validation. Upon successful password change, the session hash is updated to prevent the user from being signed out after password change, as a password change signs out all sessions for the user.

---

6 End product

During development we realized that some of our initial features were either too ambitious or less relevant, therefore we adjusted our features to be more realistic 8. The end product represents a complete application that incorporates these features. The application allow users to upload and search for songs to create personalized playlists. The design of the platform is both user-friendly and aesthetically pleasing, which gives it a strong sense of identity, it follows secure guidelines ensuring a secure user experience. Overall, we are satisfied with the end product and believe it fulfills the assignment requirements.

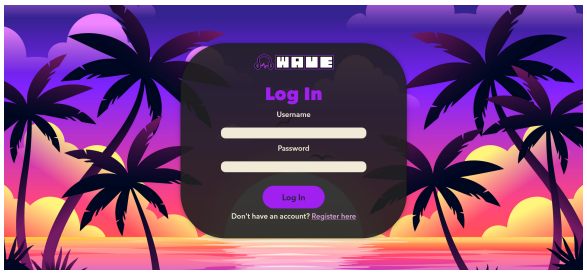
The following images display each page of the application and some of the features that are available.



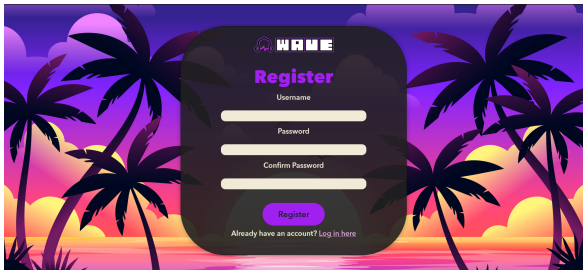
(a) Landing page.

(b) Landing page.

Figure 10: The landing page contain details about the application and prompts the user to either login or register

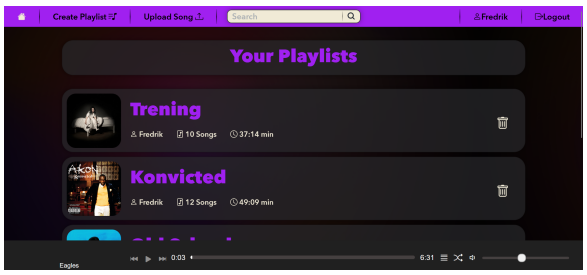


(a) Login page.

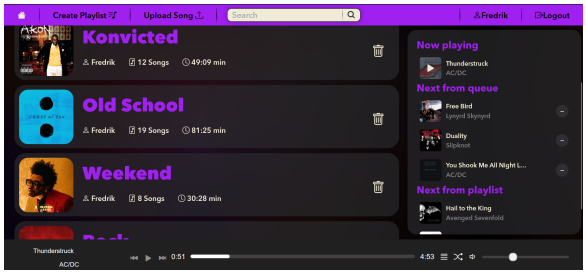


(b) Register page.

Figure 11: Users can access the application through the login or register page.

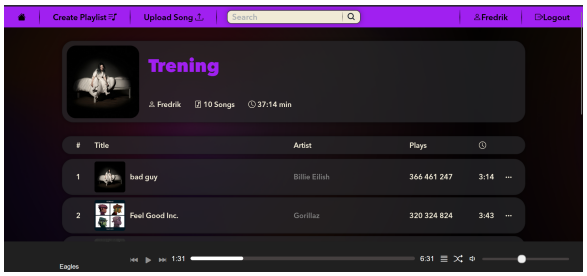


(a) Home page.

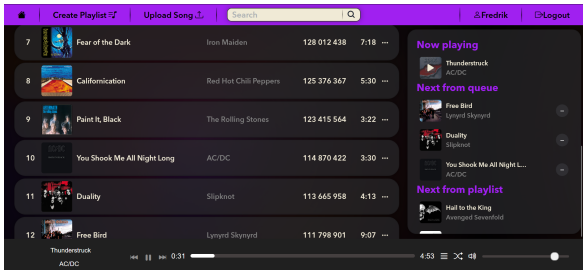


(b) Home page with queue.

Figure 12: The home page present a list of playlists for the current user.

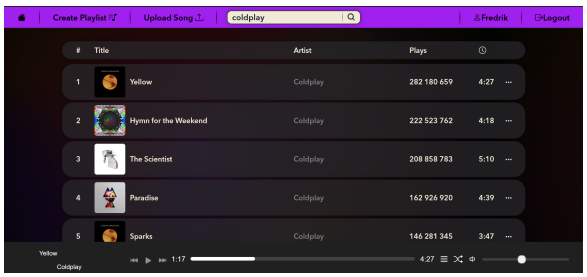


(a) Playlist page.

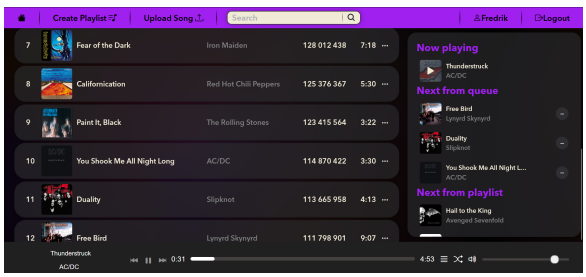


(b) Playlist page with Queue.

Figure 13: A playlist contain songs and can be accessed through the homepage.

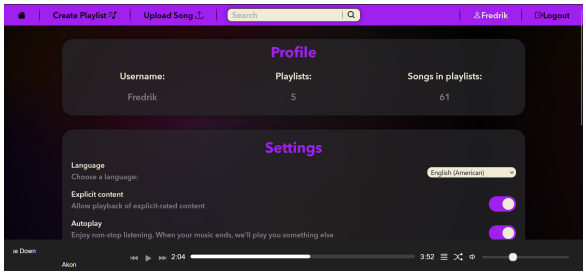


(a) Search page.

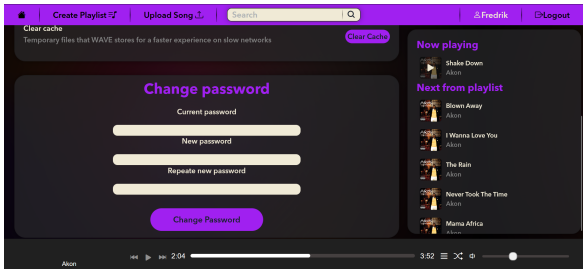


(b) Search page with Queue.

Figure 14: Users can search for songs through the navigation bar, prompting the results on the search page.

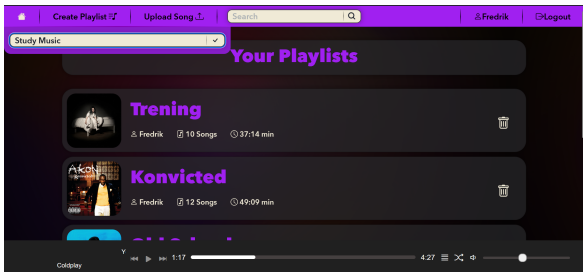


(a) Profile page.

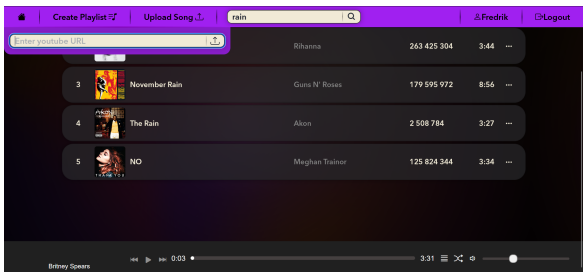


(b) Profile page with Queue.

Figure 15: Users can view details and change their password on the profile page.



(a) Create playlist.



(b) Upload song.

Figure 16: Features available through the navigation bar.

7 Discussion

7.1 Group Dynamic

7.1.1 Team cohesion

Team cohesion played a crucial role during the software development, without team cohesion, the project would not have advanced to the extent that it did. Team cohesion refers to the degree of trust, unity and camaraderie among the team members. Having spent three years studying together, we already had a sense of cohesion. Despite us not having prior experience in working together on a project of this scale, the shared experiences throughout university had already laid a good foundation for team cohesion.
Even though, there already were a strong sense of cohesion within the team, this could always be built upon. One of the main ways we strengthened team cohesion were by taking regular breaks and or doing various activities together, where we ate lunch together, went out for a couple of drinks or even played video games.

Doing non-project related activities provided opportunities for us to interact in an informal setting, facilitating personal connections beyond the project duties. Shared experiences during these activities allowed individuals to bond over common interests, thereby creating stronger team cohesion.

7.1.2 Roles and responsibilities

Having clear responsibilities within the team was paramount to the development process. By assigning each team member with specific tasks created an effective framework where each team-member could contribute effectively. However, the team only consisted of five members, thus some team-members would get multiple responsibilities.

The *Product-owner* of the team were in practice every team member, where we would think of the product as if we were the customers. As the whole team in effect were the product-owner, each member would create user stories, and in the sprint-planning the team would discuss and vote for the features that we deemed important for the end-product.

The team would regularly change which team-member were the *ScrumMaster*. Even though in a real world scenario this could be problematic as it can make the overall direction of the project unclear. This however, did not raise a problem for our team, as we had a clear vision from the beginning, and good communication throughout development. Switching up the *ScrumMaster* regularly it helped team dynamic as each member could get an understanding for the role, and its responsibilities.

7.1.3 Communication

High quality communication within the team was essential for the development process. Effective communication were maintained through the use of multiple channels. The backbone of communication within the team were informal meetings, The reasoning being that each team-member had different schedules, thus having a daily scrum meeting would prove to be time consuming. Instead of sitting through a meeting that potentially did not affect every single team member, a team member would communicate either on social media or in person if a problem arose. Every team member would be updated on Jira on who worked on which feature, the diligent use of Jira were one of the reason for using informal meetings more and only having one formal meeting a week.

Having defined roles, good team cohesion and communication, would allow the development atmosphere to be a place where the team members could trust and rely on each other, not only on showing up on time, but also in the each team members ability to solve their given assignments.

7.2 Alternative Approaches

The approach of using scrum for agile development and django/react for our application code has worked out as intended, we made the educated choice at the start and stuck with it throughout. However there are always alternative approaches, which we'll discuss in this section.

7.2.1 Scrum Alternative

As scrum is a fundamental part of agile software development and such a big part of the course syllabus, adapting another approach for this project was never an option. However, it is possible to argue for not using scrum when developing software. We will compare the scrum framework with the Kanban framework, to identify if Kanban could be a good alternative to scrum Metier (2023).

Kanban differs from scrum in two significant ways. The first is its approach to changes. In Scrum methodology, tasks must be done according to an established procedure. In Kanban, you begin with your current processes and make incremental improvements over time. Time frame is the second one. In Scrum, work is organized into time-boxed iterations called sprints, whereas in Kanban, there are no predefined time frames for completing tasks. Kanban focuses on a continuous workflow allowing for flexibility in prioritizing and completion.

The continuous flow when using Kanban would allow us to adapt to changing priorities and requirements more easily. Tasks can be added or re-prioritized at any time without waiting for the next sprint, making it ideal for environments where requirements are frequently changing or where work is unpredictable. Because of features and goals changing during development, adapting Kanban could have been a good choice. However Scrum is better suited for our needs because of its' structured approach with time-boxed iterations.

7.2.2 Language and Framework Alternatives

Alternatives to React for Front-end Development

While React is used for its component-based architecture and extensive ecosystem, there are notable alternatives such as Angular and Vue.js, each offering unique advantages. Angular provides a comprehensive framework with strong support for two-way data binding for enterprise-level applications, whereas Vue.js offers a more lightweight and intuitive approach, reducing the complexity and overhead associated with React's JSX and virtual DOM. However, we are most familiar with React, and its large community support and available libraries make it a great choice.

Alternatives to Django for Back-end Development

Django offers a robust, scalable, and secure back-end framework, however alternatives such as Flask or Express.js are viable options. Flask and Express.js is optimal for applications that benefit from a simple and flexible framework with focus on reduced overhead. Despite these options, Django's comprehensive built-in functionalities and its ability for fast development while ensuring security, make it suitable for our project. In addition we are all familiar with python, making the development process more efficient.

7.3 Lessons Learned

The development process taught us a lot about working together as team, how to communicate as a team, and how to effectively manage a project with a lot of moving pieces. Utilizing Scrum has helped us to segment the project into smaller manageable tasks and created a way for us to maintain steady development progress. However, the backlog items could be segmented even further, to make tasks even more precise.

During code reviews and debugging sessions we discovered that each team-member had their own unique coding styles, these differences would often slow progress, as we needed to explain to each-other what the code did. To mitigate this problem, we could have from the start established clear styling guidelines, which would in turn, keep the code clearer, help us debug, and to integrate our features more seamlessly. Using scrum tools such as Jira, and maintaining a well organized backlog, would prove to be invaluable, as the team could prioritize tasks, add new requirements, and also helped clarify which team member did what.

We learned however, that estimating story points could be difficult, due to our lack of experience in working with scrum. Implementing tests for the back-end from the start helped us maintain a high-level of code quality, especially later during development as user-features would utilize the back-end calls. We could however implement back-end tests even more frequently, but on the other hand, abandoning the TDD approach would help the team, since the front-end had to wait for the back-end to be complete, therefore most of the back-end tests would be written after implementation.

When using git, we created a branch for each team member, in retrospect we could and should have created a branch for each feature instead. As the feature branch method would clarify even further what feature were under development, the approach would also have made thee total git history more concise and might prevent a lot of the git issues during development. Each team member also had different experiences and standards in using git, having a defined git flow from the start, would also help.

7.4 Complexity

We think the overall complexity of our code is moderate. The code structure and organization have various levels of complexity, as some of the features demand more code to work. For instance the login page has a more simple source code than the navigation bar, which has more functionality than login.

The directory structure of the product is well organized with front- and back-end folders separating the front- and back-end code 3.3. In the front-end folder each page has dedicated child-folder containing their respective code. Same goes for back-end where there are child folders for API calls and User Authentication. Our code is well documented with comments, making it easy to understand what it does.

The most complex single functionality of the product is the music player. The music player provides all the necessary features, navigate, previous, next, pause and queue. This was the most complex functionality to implement, because of how it intertwines the features mentioned above and relies on them to function.

7.5 Problems and Choice of Solutions

We have listed up some of the problems that occurred during development, and their choice of solution:
Communication between Front-end and Back-end: In Django you have to explicitly setup settings for allowing incoming requests, which was a problem at the beginning because we did not know why our front-end couldn't talk to our back-end.
Authentication: After solving the communication problems we stumbled upon another similar problem where the front-end was never authenticated, after some debugging sessions we found out that the AxiosAxios (2024) requests did not include our authentication cookies or CSRF tokens, resulting in unauthorized access. This meant we couldn't do requests to the back-end.

---

8 Future work

As some of the initial features listed in section 2 where either too ambitious or simply not that important they are listed as future work. This section also includes features that we discovered during development, and features that would be good additions if we continued working on the project.

8.1 Listening History

Listening history would be a good addition to the application as mentioned in section 2, by recommending songs based their data, such as artist and genre. We realized we could not implement the complex algorithm that would benefit from this data, causing the feature to be dropped.

8.2 Shuffle

Also a feature listed as a goal in section 2 the shuffle functionality would give a different listening experience each time. Unfortunately it has not been implemented in due time. It was de-prioritized due to other functionalities being more important. The implementation of shuffle would not be challenging, and it would be one of the first additions added if we continued working on the product.

8.3 Audio Sources

As it stands we only support the format of Youtube URLs to upload songs. This is different from the proposed goal of the ability to upload your own mp3-files. Pytube class' Pytube (2024) access to YouTube's vast music library, along with its ease of use, made it a convenient choice during development.

If we continued on this project we would incorporate the ability to upload your own mp3-songs. This would allow users to explore and personalize their experience by further creating a whole new level of customization and creativity. We'd possibly also allow other music/video streaming platforms as sources for audio files.

8.4 Artist Page

The artist page is a feature we realized was a good addition while we were developing the product. We do support search of artist, but each artist does not have their own page. The artist page would have the artist's songs and albums. It would be beneficial for users who find a song they like to be able to discover more of the artist behind it, the artist page would incorporate this in a satisfactory way. This and shuffle would be the two first features added in the future.

8.5 Deletion of Songs

In the finished product the only way to delete uploaded songs is to open the database itself and delete the row of the database table where the song lies. This would be problematic when the database grows as more users start uploading songs. We would like to create a page showing the songs each user has uploaded, along the functionality to delete these. We would also create a page dedicated to an admin user where they have the power to delete every song in the database. This tool is great to maintain the professionalism and control of the platform, as it prevents users with malicious intent from causing harm.

---

9 Conclusion

In conclusion, we have created a well functioning web-application. Our project has successfully met many of the primary goals we set at the start of development. We were able to create an aesthetic web-based music application with the functionality to listen to your favorite songs, upload your own, and create tailored playlists. The requirement of using agile development has been done with the use of scrum. The scrum framework created a collaborative and adaptive working environment. The iterative nature of scrum enabled us to continuously improve our workflow and productivity. Overall, this project has taught us the importance of communication and flexibility when developing software. We think the final state of the product fulfills the project requirements, confirming the validity of our submission.

References

Axios, C. (2024). Axios documentation. Accessed: May 14, 2024.

Metier (2023). Hva er kanban? Accessed: May 13, 2024.

Pytube, C. (2024). Pytube documentation. Accessed: May 12, 2024.

Sommerville, I. (2021). *Engineering Software Products. An Introduction to Modern Software Engineering.* Pearson.

10 Appendix

10.1 A User Stories

User story 1: As a user, I want a personalized home page so that I can see my favorite tracks and recommended new songs.

User story 2: As a visitor, I want a straightforward login page to easily access my account.

User story 3: As a user I should be able to view my profile and change user specific settings like my password.

User story 4: As a music enthusiast, I want to be able to create and view playlists so that I can categorize songs for different moods and occasions.

User story 5: As a user, I want a consistent header and footer bar for easy navigation through the app.

User story 6: As a listener, I want a song search feature to find my favorite tracks quickly.

User story 7: As a listener, I want a user-friendly music player with basic controls like play, pause, and skip so that I can enjoy uninterrupted music.

User story 8: As an artist i want to be able to upload my own songs to the application so other users can discover and share my music.

User story 9: As a user, i want seamless routing between pages to improve my experience using the app.

User story 10:: As a user i want to be able to queue songs from different playlists in case.

User story 11:: As a user i want a secure authentication system to protect my personal information and listening data.

User story 12:: As a listener, I want to have essential controls like play, pause, skip, and volume adjustment so that I can easily manage my music playback according to my preferences.

User story 13:: As a listener, I want a user-friendly music player interface that is visually appealing and easy to navigate so that I can have a pleasant and intuitive music listening experience.

10.2 B Personas

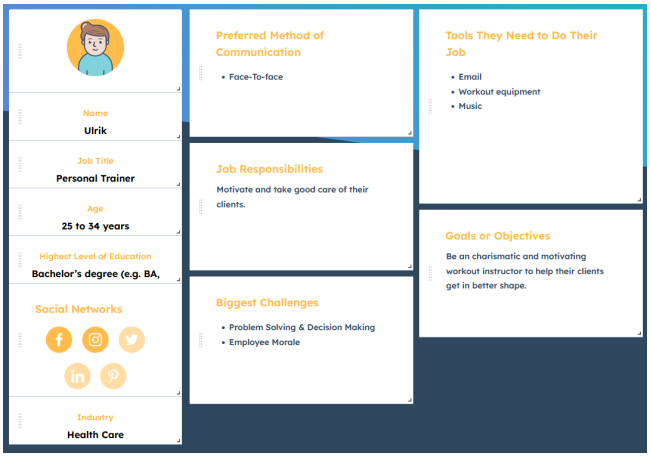


Figure 17: Personas #1

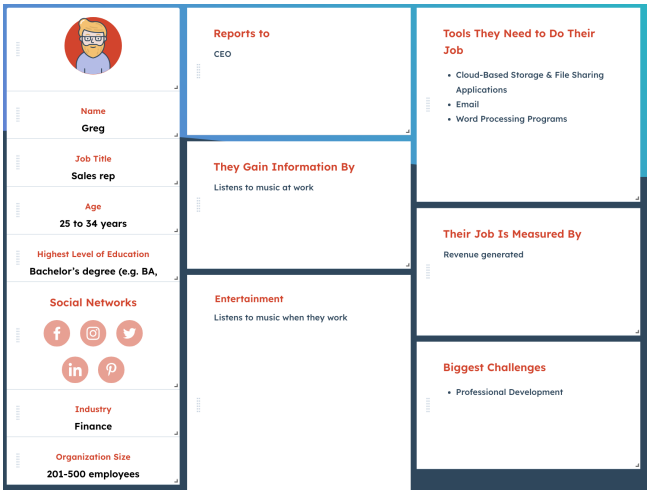


Figure 18: Personas #2

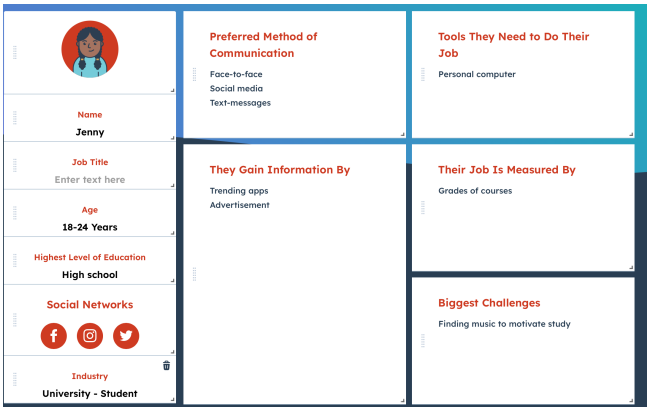


Figure 19: Personas #3

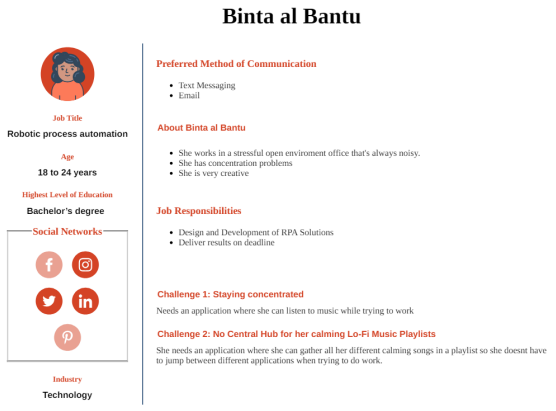


Figure 20: Personas #4

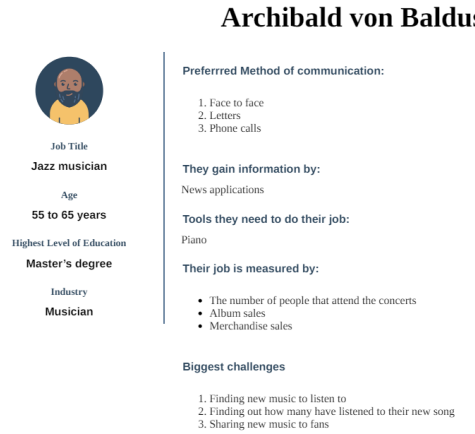


Figure 21: Personas #5

10.3 C Scenarios

10.3.1 Scenario 1

Binta al Bantu, has no platform where she can put all her obscure LoFi music, she discovered Wave, where she can upload and centralize all her Lofi songs. 20

10.3.2 Scenario 2

Archibald von Baldus, known to his friends as Archie B, is a Jazz musician. Archie B would like to upload his music frequently so that his colleagues can give him quick feedback. "Damn these music labels and music platforms!" Archie B yelled. In a furious rage, and a search for a better solution Archie B discovered Wave, which had an easy upload button that let him instantly share his latest musical compositions with his peers. in good ol' Jazz fashion Archie B yelled "Skibidabado" and smiled. 21

10.4 D Contribution

10.4.1 Individual contributions

- Amund Strøm
  - Designed the User Interface, and created the visual assets.
  - Implemented front-end components using React and SCSS.
  - Ensured consistency in design and user experience across the website.
- Iver Mortensen
  - Responsible for website security.
  - Created the login and register functionality.
  - Implemented back-end calls and test coverage using python.
- Marcus Ryan
  - Designing the front-end architecture.
  - Implemented front-end components using React and SCSS, such as the playlist page.
  - Reported and handled bugs, ensuring a high quality code base.
- Tarek Lein
  - Designed the database schema and wrote back-end API calls.

  - Extensive back-end testing.
  - Provided valuable feedback in sprint review meetings, focusing on improving code quality and test coverage.

- Theodor Tredal
  - Implemented front-end components using React and SCSS.
  - Main responsibility for implemented the music player, and footer component.
  - Reported bugs and updated the backlog on Jira.

10.4.2 Collective contribution

The project would not have gotten as far as it did without collective contribution and the use of pair programming.
Here are some examples of pair programming during development.

- Back-end architecture and initial implementation - Tarek Lein and Iver Mortensen.
- - Music player and footer functionalities - Theodor Tredal and Tarek Lein.
- - Deletion of playlists - Amund Strom and Marcus Ryan.

10.4.3 Special mentions

The contribution of our TA, Sofie Johansen, were proven to be invaluable, as she were an mentor and helped guide the team in the right direction during the development process.

10.5 E Backlog (Tarek)

10.5.1 Sprint 1

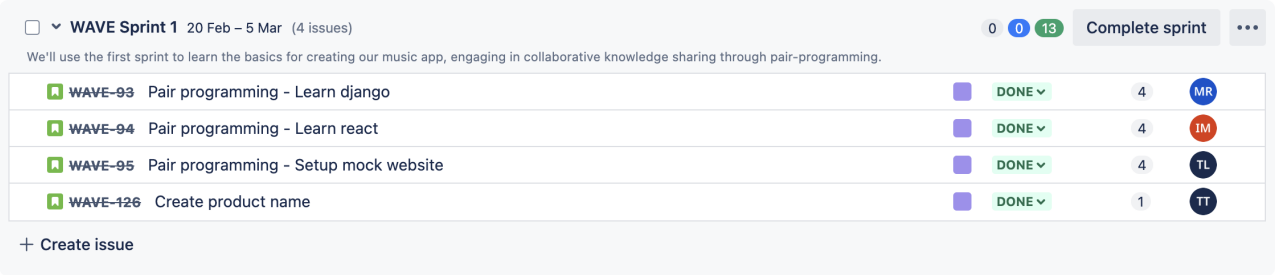
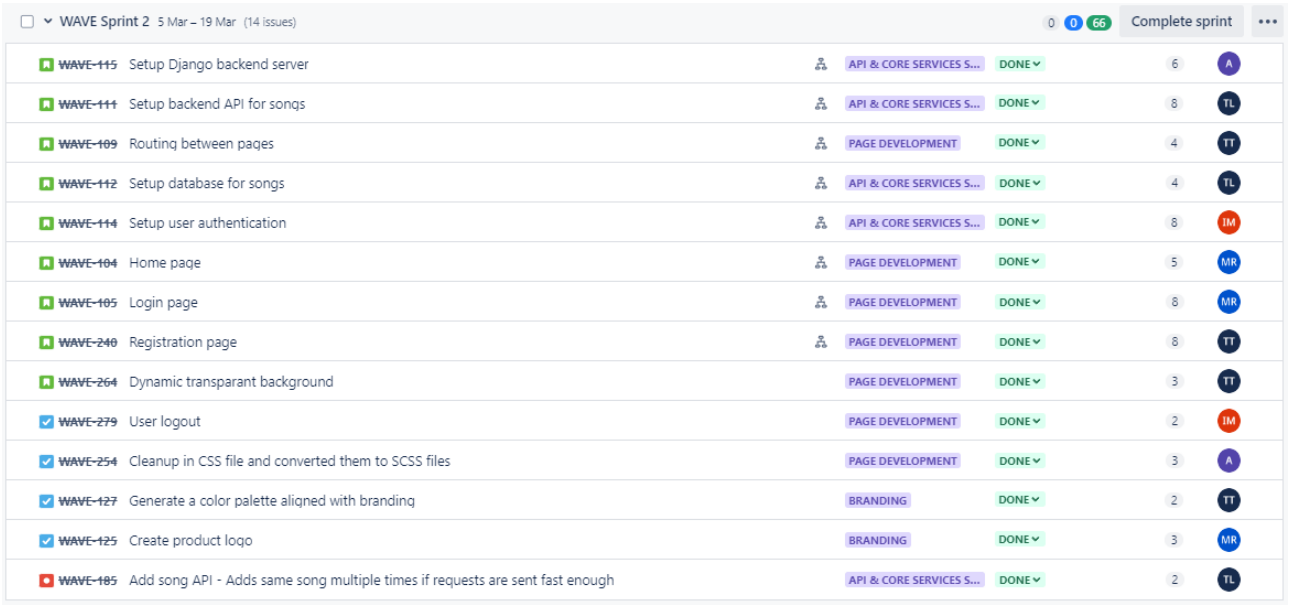


Figure 22: Sprint 1 full picture

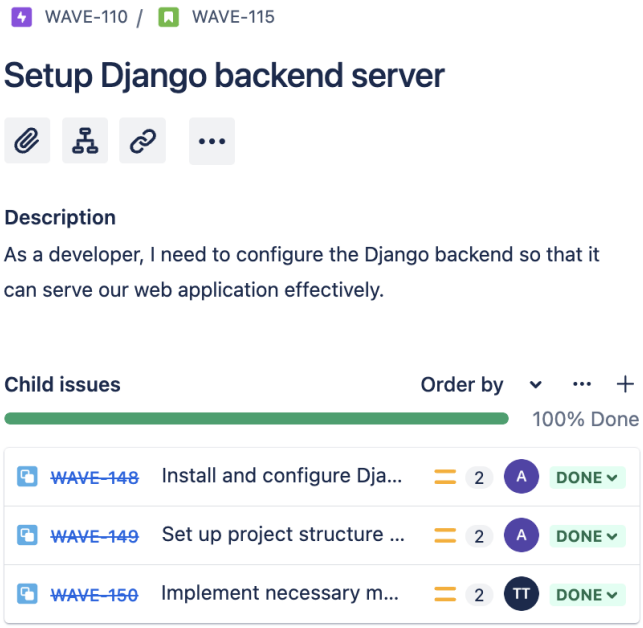
10.5.2 Sprint 2



WAVE Sprint 2 5 Mar – 19 Mar (14 issues) — 0 0 66 — Complete sprint

~~WAVE-115~~	Setup Django backend server	API & CORE SERVICES S...	DONE	6
~~WAVE-111~~	Setup backend API for songs	API & CORE SERVICES S...	DONE	8
~~WAVE-109~~	Routing between pages	PAGE DEVELOPMENT	DONE	4
~~WAVE-112~~	Setup database for songs	API & CORE SERVICES S...	DONE	4
~~WAVE-114~~	Setup user authentication	API & CORE SERVICES S...	DONE	8
~~WAVE-104~~	Home page	PAGE DEVELOPMENT	DONE	5
~~WAVE-105~~	Login page	PAGE DEVELOPMENT	DONE	8
~~WAVE-240~~	Registration page	PAGE DEVELOPMENT	DONE	8
~~WAVE-264~~	Dynamic transparant background	PAGE DEVELOPMENT	DONE	3
~~WAVE-279~~	User logout	PAGE DEVELOPMENT	DONE	2
~~WAVE-254~~	Cleanup in CSS file and converted them to SCSS files	PAGE DEVELOPMENT	DONE	3
~~WAVE-127~~	Generate a color palette aligned with branding	BRANDING	DONE	2
~~WAVE-125~~	Create product logo	BRANDING	DONE	3
~~WAVE-185~~	Add song API - Adds same song multiple times if requests are sent fast enough	API & CORE SERVICES S...	DONE	2

Figure 23: Sprint 2 full picture



WAVE-110 / WAVE-115

Setup Django backend server

Description

As a developer, I need to configure the Django backend so that it can serve our web application effectively.

Child issues — Order by — 100% Done

~~WAVE-148~~	Install and configure Dja...	2	A	DONE
~~WAVE-149~~	Set up project structure ...	2	A	DONE
~~WAVE-150~~	Implement necessary m...	2	TT	DONE

Figure 24: Sprint 2 - 1. story

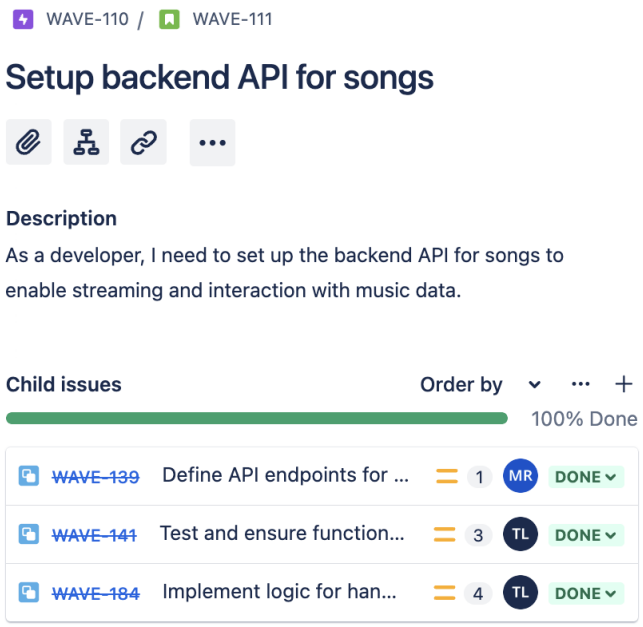


Figure 25: Sprint 2 - 2. story

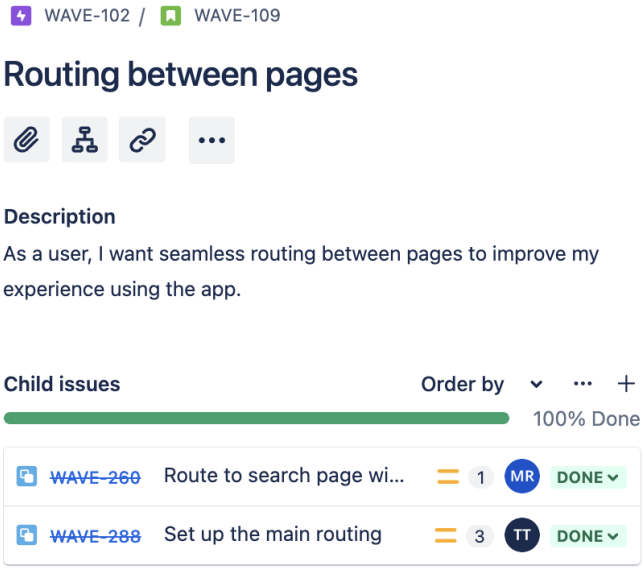


Figure 26: Sprint 2 - 3. story

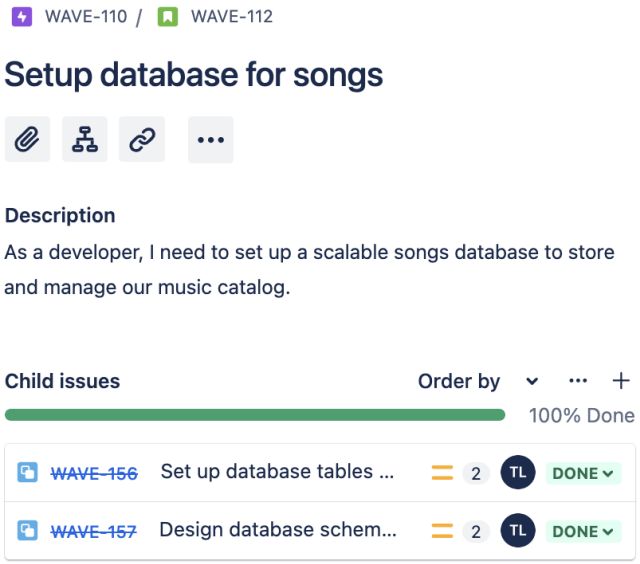


Figure 27: Sprint 2 - 4. story

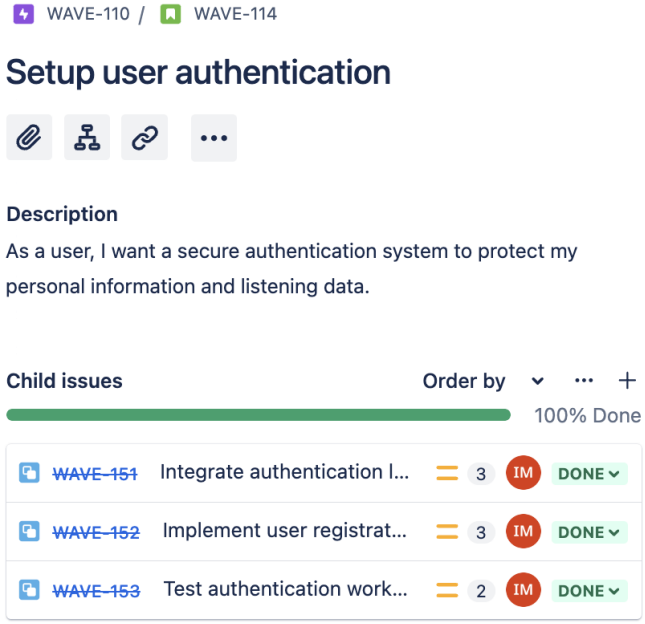


Figure 28: Sprint 2 - 5. story

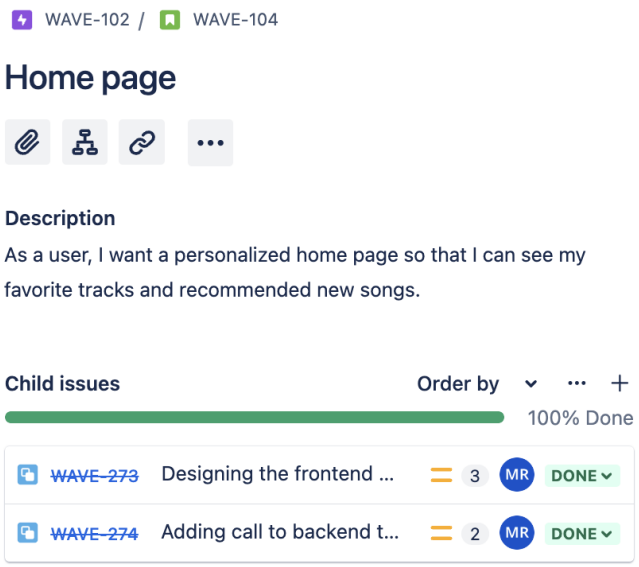


Figure 29: Sprint 2 - 6. story

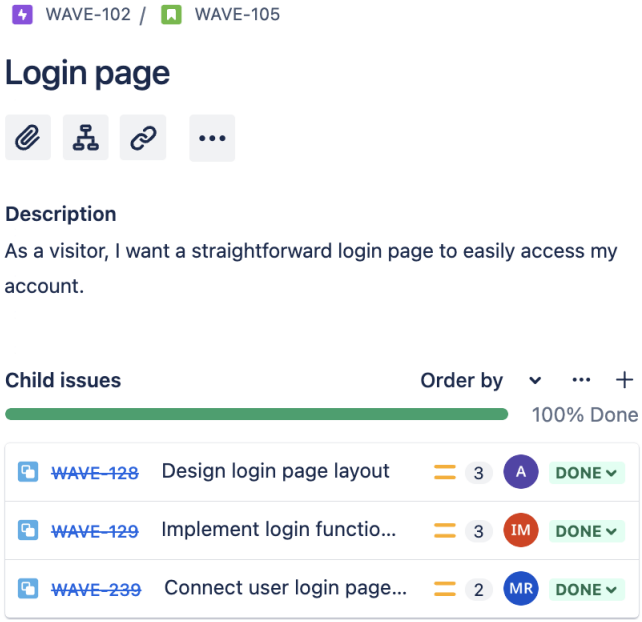


Figure 30: Sprint 2 - 7. story

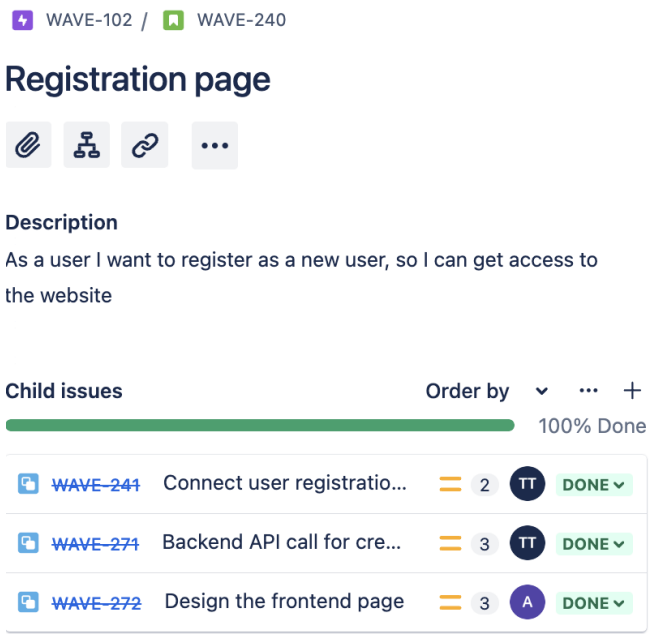


Figure 31: Sprint 2 - 8. story

10.5.3 Sprint 3

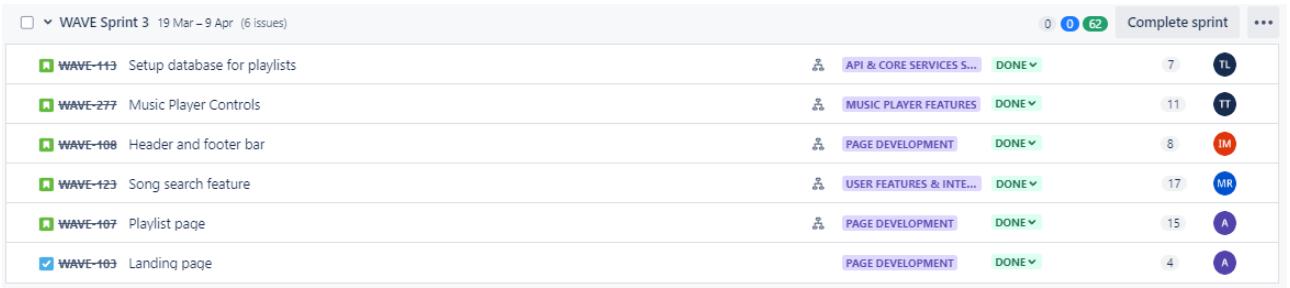


Figure 32: Sprint 3 full picture

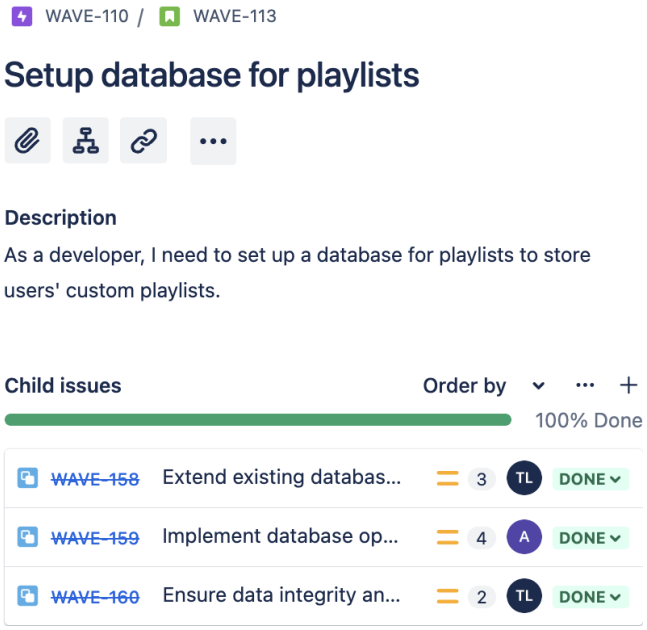


Figure 33: Sprint 3 - 1. story

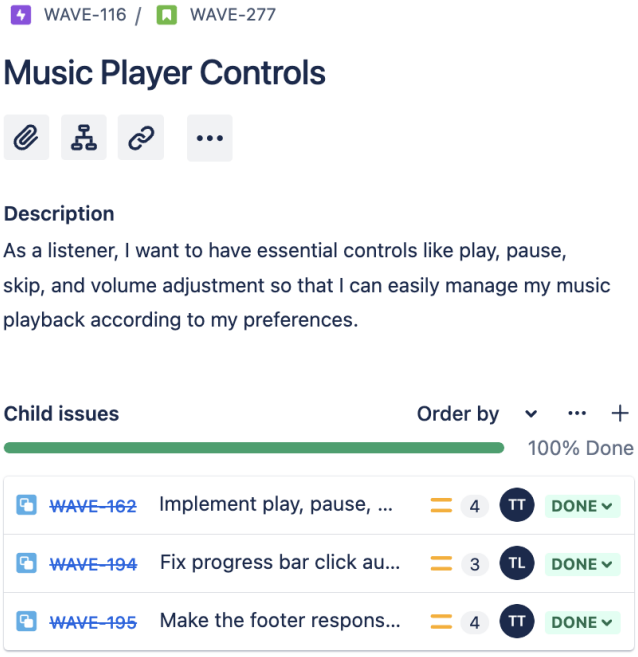


Figure 34: Sprint 3 - 2. story

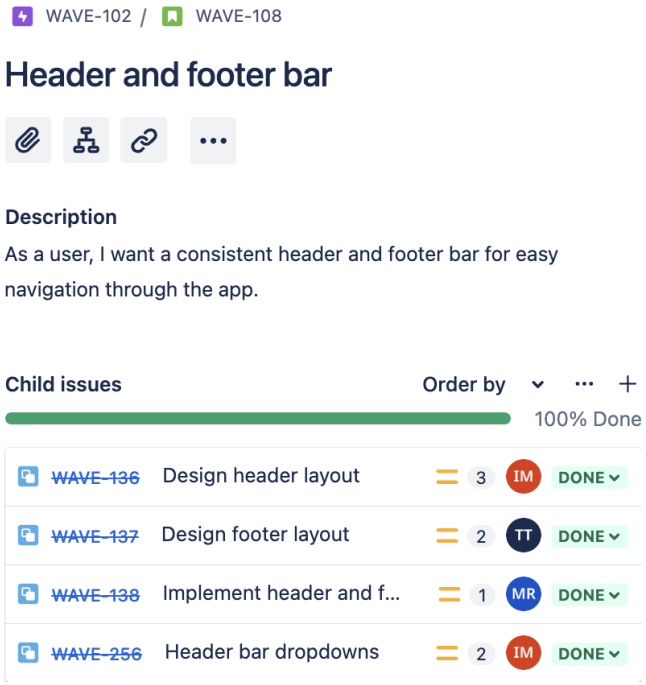


Figure 35: Sprint 3 - 3. story

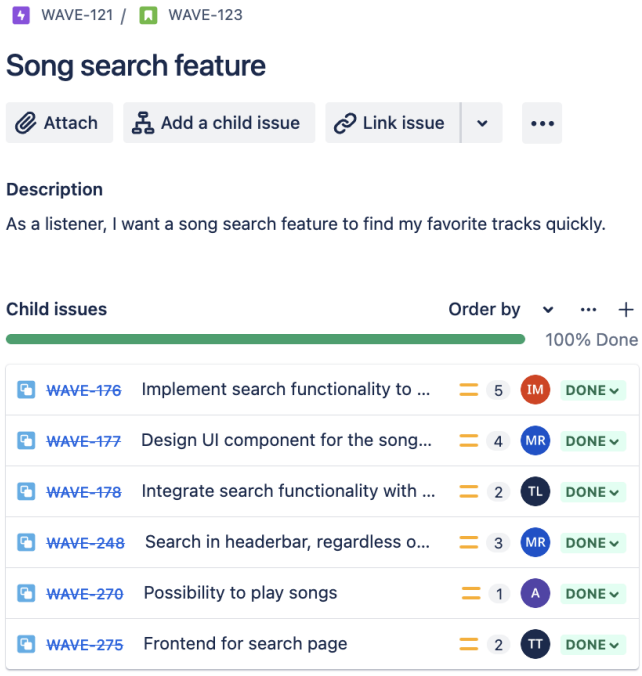


Figure 36: Sprint 3 - 4. story

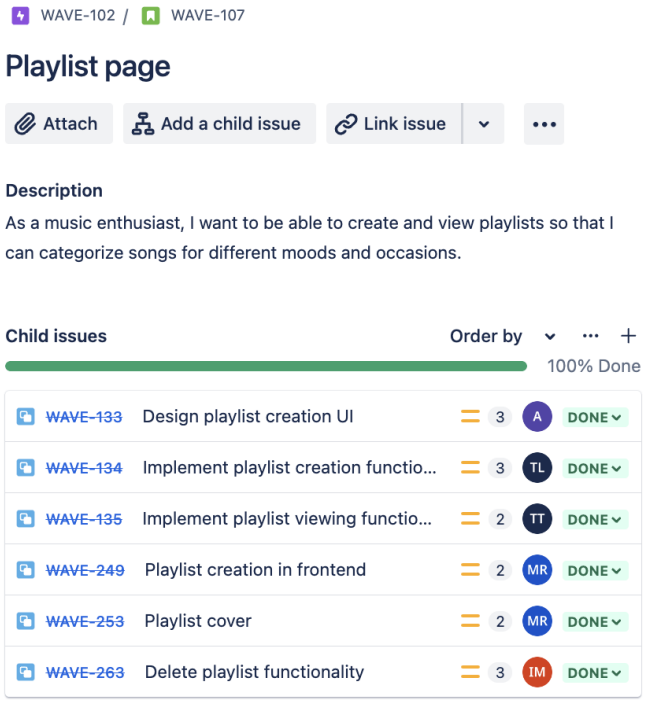


Figure 37: Sprint 3 - 5. story

10.5.4 Sprint 4

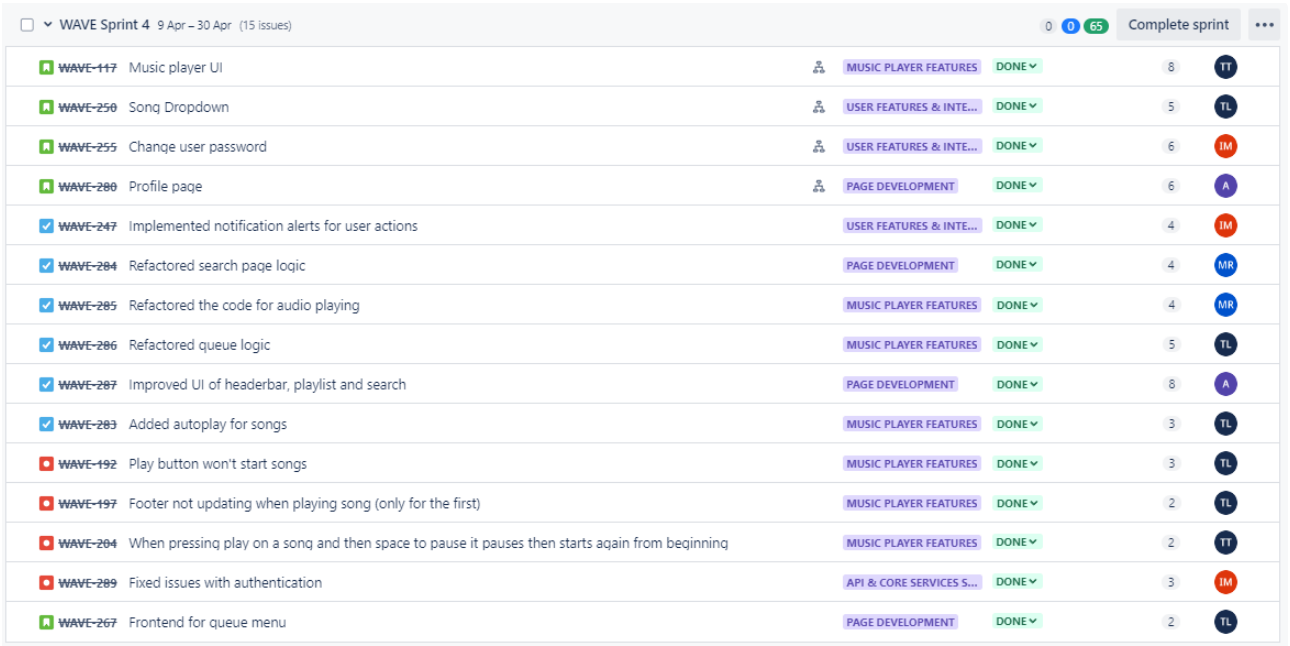


Figure 38: Sprint 4 full picture

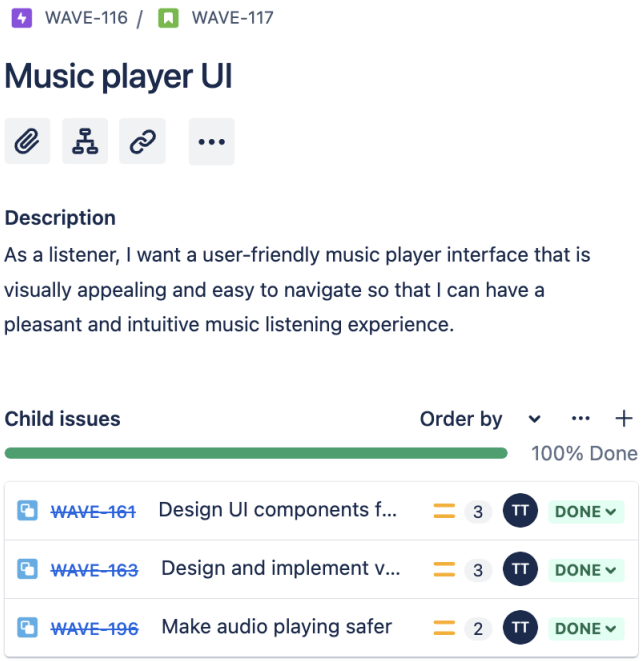


Figure 39: Sprint 4 - 1. story

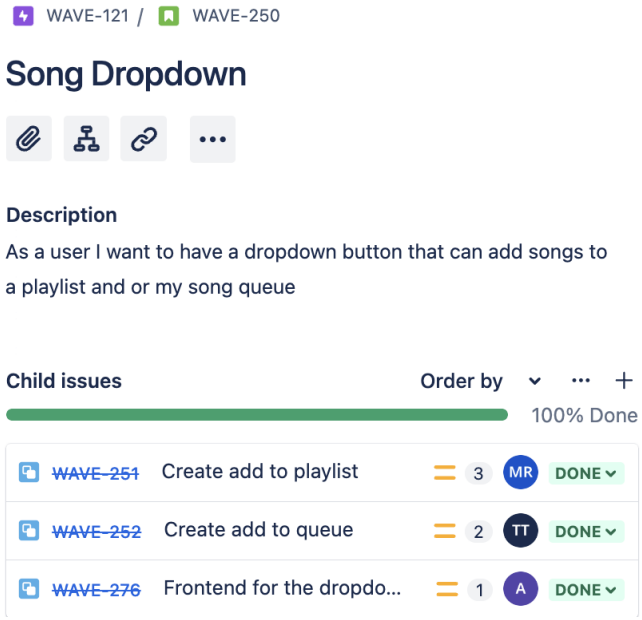


Figure 40: Sprint 4 - 2. story

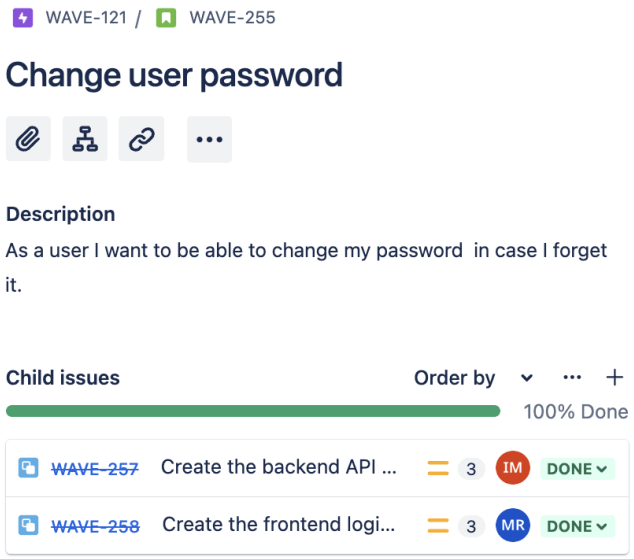


Figure 41: Sprint 4 - 3. story

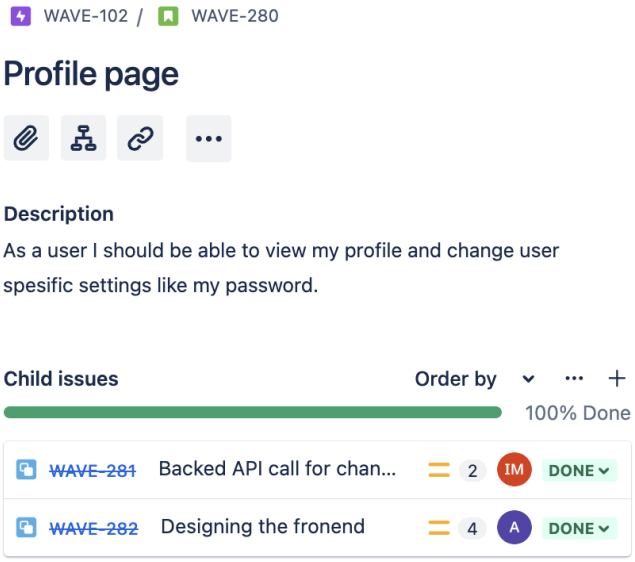


Figure 42: Sprint 4 - 4. story

10.5.5 Remanding backlog

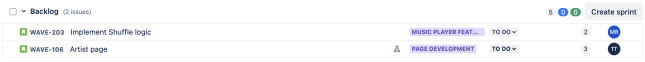


Figure 43: Remanding items in the backlog

- Insert pictures of the backlog.
- Make sure that the pictures include story points. This is important as time estimation is a requirement.

10.6 F Meeting Notes

Meeting: January 30 (Pre-Sprint Planning)

- **Ideas Discussed:**
  - Spotify-like application
  - Messaging forum
  - Fitness app
  - Calorie calculator
- **Progress:**
  - Started setting up Django
- **Action Items for Next Meeting:**
  - Write product vision, user stories, scenarios, personas

Meeting: February 13 (Pre-Sprint Planning)

- **Progress:**
  - User stories written
  - Personas created
  - Feature list compiled
- **Decision:** Opt to develop the Spotify-like application
- **Features Planned:**
  - Search for songs
  - Create playlists
  - Upload MP3 files (admin feature potentially)
  - History and queue features
  - View other's playlists (general user role)
- **Preparation:** Set up backlog for the first sprint

Meeting: February 20 (Pre-Sprint Planning)

- **Progress:**
  - Set up backlog in Jira; discussed breaking tasks into smaller tasks
  - Emphasized the importance of testing before progressing to next tasks

Meeting: February 20 (Sprint - 1 Kickoff)

- **Sprint Duration:** Two weeks (February 20 - March 5)
- **Roles:** Scrum master - Tarek
- **Focus:** Login/homepage, database song entries

Meeting: February 27 (Sprint - 1)

- **Notes:** Limited progress due to exams
- **Focus:** Login/homepage, database song entries
- **Progress:** –

Meeting: March 5 (Sprint - 1 Review)

- **Notes:**
  - Jira backlog reviewed; needs better task segmentation
- **Progress:**
  - Login page completed; integration with database pending
  - Visualization of the database in Miro
- **Retrospective:**
  - **Positives:**
    - Managed to make progress despite exams
  - **Improvements:**
    - Clarify tasks in backlog, meet more frequently

Meeting: March 5 (Sprint - 2 Kickoff)

- **Sprint Duration:** Two weeks (March 5 - March 19)
- **Roles:** Scrum master - Amund
- **Focus:** Login/homepage, database song entries

Meeting: March 12 (Sprint - 2)

- **Notes:** Backlog refined, tasks detailed
- **Focus:** Styling, color palette, logo, database integration
- **Progress:**
  - Setup Django back-end server
  - Setup database for songs
  - Setup back-end API for songs to be able to interact with the data

Meeting: March 19 (Sprint - 2 Review)

- **Progress:**
  - Color palette and product logo created
  - Login- & registration-page implemented
  - User authentication & logout implemented
  - Homepage implemented
- **Retrospective:**
  - **Positives:**
    - Effective use of pair programming; back-end (Tarek and Iver), front-end (others)
  - **Improvements:**
    - Regular updates among team members on code changes

Meeting: March 19 (Sprint - 3 Kickoff)

- **Sprint Duration:** Three weeks (March 19 - April 9)
- **Roles:** Scrum master - Iver
- **Focus:** front-end and back-end integration
- **Notes:** Next meeting April 9th, due to easter break

Meeting: April 9 (Sprint - 3 Review)

- **Progress:**
  - Setup database for playlists
  - playlist page implemented; creation and management
  - Setup music player
  - Song search implemented
  - User account management
  - 22 tests for API calls
- **Retrospective:**
  - **Positives:**
    - Work accomplished
    - Improved use of Jira for task detailing
  - **Improvements:**
    - More frequent status meetings

Meeting: April 9 (Sprint - 4 Kickoff)

- **Sprint Duration:** Three weeks (April 9 - April 30)
- **Roles:** Scrum master - Marcus
- **Focus:** front-end styling, coverage on API tests

Meeting: April 16 (Sprint - 4)

- **Notes:** Meeting cancelled due to illness
- **Focus:** –
- **Progress:** –

Meeting: April 23 (Sprint - 4)

- **Notes:** Product demo held; product looking polished and well-executed
- **Focus:** Polishing the code and adding finishing touches to complete the project
- **Progress:**
  - Music player UI and controls
  - Profile page implemented
  - Improved UI for headerbar, playlist- and search-page
  - Adding songs to queue and playlists
  - Front-end for the queue

Meeting: April 30 (Sprint - 4 Review)

- **Progress:**
  - Password change feature added, needs back-end linking
  - Playlist modification features developed
  - Notification alerts for user actions
  - Refactored code
- **Retrospective:**
  - **Positives:**
    * Significant UI enhancements and application polishing
    * Alot of pair programming, worked together
    * Talked together about what we have accomplished
  - **Improvements:**
    * Regular Jira updates needed