Group symbol: **W04IST-SI4023P-3**

Team: **3.3**

Project title: **TaleTinker**

Team members (*filled by PM, Team Leader*):

No	Name	Surname	Student ID	Role
1	Mateusz	Molenda	259905	*PM, Team Leader*
2	Abrorjon	Ruziboev	269617	*Team member*
3	Maurycy	Jakiel	266622	*Team member*
4	Dominika	Kołowrotkiewicz	266908	*Team member*

4. Construction and tests (**F4**)

4.1. Implementation

Backend artifacts:

```
├── alembic.ini
├── docker-compose.yml
├── docker_deploy.sh
├── Dockerfile
├── init.sql
├── poetry.lock
├── pyproject.toml
├── README.md
├── scripts
│   └── entrypoint.sh
├── src
│   ├── api
│   │   ├── __init__.py
│   │   └── v1
│   │       ├── endpoints
│   │       │   ├── account.py
│   │       │   ├── audiobook.py
│   │       │   ├── category.py
│   │       │   ├── chapter.py
│   │       │   ├── __init__.py
│   │       │   ├── listening.py
│   │       │   ├── review.py
│   │       │   ├── settings.py
│   │       │   └── user_audiobook.py
│   │       └── __init__.py
│   ├── app.py
│   ├── core
│   │   ├── auth.py
```

```
|   |   ├── errors.py
|   |   └── __init__.py
|   ├── db
|   |   ├── builder.py
|   |   ├── decorators.py
|   |   ├── __init__.py
|   |   ├── migrations
|   |   |   ├── env.py
|   |   |   ├── README
|   |   |   ├── script.py.mako
|   |   |   └── versions
|   |   |       ├── 08473aef612a_chapter_and_user_chapter.py
|   |   |       ├── 1347aa4629a1_default_to_server_default.py
|   |   |       ├── 5f6bbb80201c_initial_revision.py
|   |   |       ├── 7c70c2bca38f_server_default_to_default.py
|   |   |       ├── 8b5afc6fb3e6_user_audiobook_added.py
|   |   |       └── e8fff70a5889_review_and_listening_tables.py
|   |   └── models.py
|   ├── __init__.py
|   ├── repo
|   |   ├── account.py
|   |   ├── audiobook.py
|   |   ├── base.py
|   |   ├── category.py
|   |   ├── chapter.py
|   |   ├── __init__.py
|   |   ├── listening.py
|   |   ├── review.py
|   |   ├── user_audiobook.py
|   |   ├── user_chapter.py
|   |   └── user_settings.py
|   ├── schemas
|   |   ├── account.py
|   |   ├── audiobook.py
|   |   ├── base.py
```

```
│   │   ├── category.py
│   │   ├── chapter.py
│   │   ├── __init__.py
│   │   ├── listening.py
│   │   ├── review.py
│   │   ├── user_audiobook.py
│   │   ├── user_chapter.py
│   │   └── user_settings.py
│   ├── settings.py
│   └── web
│       ├── admin.py
│       ├── auth.py
│       ├── factory.py
│       └── __init__.py
└── tests
    ├── conftest.py
    └── __init__.py
```

16 directories, 70 files total

Dockerfile (file for creating docker image serving as server)

src (folder with source code for the application to run)

src/api/__init__.py (code for initiating the router)

src/api/v1/__init__.py (code for initiating router for managing routes defined in endpoints below)

src/api/v1/endpoints: (files defining the endpoints for the FastApi REST server, each file is managing different part of objects/tables, split allows for easier debugging and future management)

src/db/migrations/versions (folder containing database migrations)

src/db/migrations/__inti__.py (file for initiating database session)

src/db/migrations/builder.py (defining building database engine, later used)

src/db/migrations/models.py (main file containing definition for each table int the database, along with relationships between them)

src/app.py (running the application builded from src/web/factory.py

src/settings.py (defining type of database to use [here Postgres], which port for database connection, api_key for development communication with mobile app, defining what user connects to database, defining database pool size, etc. general settings to be used by database and application in general)

src/repo/ (folder containing repo files/classes used as an intermediary for communication between api logic layer to database objects, each repo class additionally has validation schema to make sure the data is in correct form, used as output format for api calls defined in src/api/v1/endpoints)

src/repo/base.py (defining the BaseRepo class)

src/repo/ 'etc.'.py (other classes for database objects inheriting from base repo class)

src/schemas (folder for schema files, used in repo class for validating the output)

Frontend artifacts:

```
696 directories, 1085 files
```

which is too much to show as earlier, as it would take over 80 pages. Most of the files are fonts, or files specific for android, IOS, Windows etc. as the app was developed using flutter, so that it can run on many OS. So here are files written by us:

```
├── lib
|   ├── config
|   |   ├── base_url_config.dart
|   |   └── constant_config.dart
|   ├── data
|   |   ├── datasource
|   |   |   ├── audiobook_chapters_remote_data_source.dart
|   |   |   ├── audiobook_remote_data_source.dart
|   |   |   ├── category_remote_data_source.dart
|   |   |   └── settings_remote_datasource.dart
|   |   ├── model
|   |   |   ├── audiobook.dart
```

```
|   |   |   ├── audiobook_listening.dart
|   |   |   ├── category.dart
|   |   |   ├── settings.dart
|   |   |   └── user_audiobook.dart
|   |   └── repository
|   |       ├── audiobook_chapters_repository.dart
|   |       ├── audiobook_repository.dart
|   |       ├── category_repository.dart
|   |       └── settings_repository.dart
|   ├── error
|   |   ├── exceptions.dart
|   |   └── failure.dart
|   ├── main.dart
|   ├── pages
|   |   ├── audiobook.dart
|   |   ├── catalogue.dart
|   |   ├── category.dart
|   |   ├── home.dart
|   |   ├── listening.dart
|   |   ├── login.dart
|   |   ├── search.dart
|   |   ├── settings.dart
|   |   ├── shelf.dart
|   |   └── signup.dart
|   ├── providers
|   |   ├── bottom_sheet.dart
|   |   ├── listening_session.dart
|   |   └── user_settings.dart
|   ├── routes.dart
|   ├── services
|   |   ├── network_info.dart
|   |   ├── network_service.dart
|   |   ├── network_service_response.dart
|   |   └── restclient.dart
|   ├── themes
```

```
|   |   └── main_theme.dart
|   ├── utils
|   |   ├── audio_player_manager.dart
|   |   └── uidata.dart
|   └── widgets
|       ├── base_tab_bar.dart
|       ├── content_item.dart
|       ├── content_section.dart
|       ├── listening
|       |   ├── choice_button.dart
|       |   ├── controls_hero.dart
|       |   ├── listening_controls.dart
|       |   ├── path_choices.dart
|       |   ├── progress_hero.dart
|       |   └── story_visualizer.dart
|       ├── promo_banner.dart
|       └── section_header.dart
```

lib/config/configs (development bool flag, files defining development and production servers to connect to along with API key to use while in production)

lib/data/datasource/data_sources (methods for api calls to the server to retrieve information about audiobooks)

lib/data/model/models (class defining actual objects such as Audiobook used in the application)

lib/data/repository/repos (classes combining models and network information [not used for now, still in development])

lib/error/errors (definitions of errors/exceptions used in the application)

lib/pages/pages (main code defining how each page should behave and be rendered on the screen)

lib/providers/providers (providers used in the application - objects used to preserve and share state among the whole application while changing pages, used by pages to check and change global state of application)

lib/services/. . . (services used by other methods, mostly by repositories, restclient.py is used by data_sources to manage the way they communicate with rest server)

lib/themes/main_theme.dart (main theme of an application, used mainly in pages whenever there's something to render on the screen, the colors and font style will be used directly from main_theme)

lib/utils/uidata.dart (definitions for routing among pages, defining what routes should lead to which pages, const configuration)

lib/utils/audio_player_manager.dart (defining class to handle logic about connecting and managing data stream when downloading/streaming audiobook)

lib/widgets/. . . (custom widgets used in the application to produce repeatable same results, widgets for display audiobook)

lib/widgets/listening/. . . (widgets specific to listening displays)

lib/routes.dart (defining the router which manages actual logic behind switching pages, and how they should be built)

lib/main.dart (initial starting point of an application)

pubsec.yaml (file with requirements for flutter project,

specify version of flutter, dart sdk, along with libraries needed to run and compile the application)

```
version: 1.0.0+1
environment:
    sdk: '>=3.1.5 <4.0.0'
dependencies:
    flutter:
    sdk: flutter
```

```
    cupertino_icons: ^1.0.2
    go_router: ^12.1.1
    audio_video_progress_bar: ^2.0.1
    provider: ^6.1.1
    http: ^1.1.2
    internet_connection_checker: ^1.0.0+1
    dartz: ^0.10.1
    equatable: ^2.0.5
    lorem_ipsum: ^0.0.3
    just_audio: ^0.9.36
    just_audio_mpv: ^0.1.7
    rxdart: ^0.27.7
    flutter_rating_bar: ^4.0.1
    auto_size_text: ^3.0.0
    connectivity: ^3.0.6
    shared_preferences: ^2.0.10
dev_dependencies:
    flutter_test:
    sdk: flutter
    flutter_lints: ^2.0.0


flutter:
    uses-material-design: false
    assets:
    - images/
    fonts:
    - family: SF Pro
    fonts:
    - asset: fonts/SF-Pro.ttf
    - asset: fonts/SF-Pro-Italic.ttf
        style: italic
```

4.2. Tests

The documentation should cover the methods and test results of the specific system's element both from the functional and non-functional point of view. The scope and the description

should conform to the specification of the requirements (F2). You should include a description of at least one test of each type of requirement (e.g. business logic, user interface, data exchange, etc.).

4.2.1. Requirements Tests

In this section, you should differentiate the description of tests from manual and automated tests. You should include scripts or test case scenarios in the report.

4.2.1.1. Functional requirements tests

(User interface) R2

[The user should be able to browse audiobooks in the store or in his own library. These stories should cover various genres such as fiction, non-fiction, mystery, romance, science fiction, etc.]. Manual testing by checking if audiobooks in home, shelf and category pages show up correctly and can be played.

(Data exchange) R4

[Users should have the ability to rate audiobooks.]. Manual testing by checking if after the audiobook is rated, a given user rating is shown on the next opening of the page, as well as if the general rating of the audiobook changes.

(Business logic) R8

[The user should be able to browse audiobooks in the store or in his own library. These stories should cover various genres such as fiction, non-fiction, mystery, romance, science fiction, etc.]. Manual testing by checking if after uploading a new audiobook it can be found in the search page.

4.2.1.2. Non-Functional requirements tests

(Security) R9

[Users must be able to create an account, log in and out of their account]. Manual testing by checking if after uploading a new audiobook it can be found in the search page.

(Reliability) R12

[In case of an internal error there should exist recovery plan /procedures to minimize downtime and potential data loss]. Manual testing, save server state to external device and forcibly shut it down, deleting all the data. Then restart server and check if data data loaded from the backup server has loaded correctly and is the same as the one saved just before shutdown.

(Maintainability) R17

[At least 90% of all methods/classes should be documented]. Automated testing using python ‘pylint’ module which allows checking the percentage of documented classes and functions in given python files. And for dart files, manual testing.

(Localization) R18

[The application should support multiple languages (at least top 3 most used by the (target) client base)]. Manual testing by checking if language is properly implemented and everything is properly translated.

(Usability) R19

[The platform should have a library of audiobooks available for users to listen to, at least 20 different audiobooks]. Automated testing by checking the server response length in the application when requesting for all available audiobooks. Simple ‘response.length < 20’ check.

4.2.2. Remaining Tests