Test strategy PLEX Tool

[Which functionality should be tested? 1](#_Toc767636371)

[Backend 1](#_Toc1484733111)

[Frontend 1](#_Toc701269909)

[Which code will not be tested and why? 2](#_Toc1636523331)

[What type of testing is used and why? 2](#_Toc1346974528)

[Which test tooling will be used 2](#_Toc1397243970)

# Which user stories should be tested and why?

Ideally every user story will be tested. And the Product Owner should only accept user stories which have been tested thoroughly. However, this is in practice often not the case, because of lack of expertise, varying skill-levels of team members and time constraints. Therefore, only the user stories which meet the Definition of Done and have been accepted by the Product Owner must be tested.

The following user stories have been accepted and thus will be tested

* As a Project Leader, I want to view the projects, which are available on the Dex platform, in the Plex application. So that, I can easily identify which projects I want to include in a playlist.
* As a Project Leader, I want to add projects to a playlist and save the playlist in the Plex application. So that, I can share the play list with my students

# Which functionality should be tested?

Ideally you would want to test everything to perfection, but this is not worth the time and effort. Therefore, we have identified the most important parts of the system to test. These parts provide essential functionality and thus must be tested.

## Backend

The backend exposes an API. The endpoints of this API must function correctly, testing these endpoints should include testing if the URL of the endpoint adheres to its specification, testing if the endpoint accepts the specified HTTP request, testing if the endpoints respond with the specified HTTP response, testing if a HTTP response contains its specified resource in the response body (if applicable).

API controllers should also be tested, this should include testing if the validation of the resource sent in the body of a HTTP request works as specified. We rely on the AutoMapper library to map our domain entities from and to resources. These mappings should be tested to ensure that the configuration of AutoMapper is done correctly. Furthermore, the services in the backend core project should be tested. As well as integration with the MSSQL database.

## Frontend

To function correctly, the frontend depends on the API of the backend and of the DeX API. So, every HTTP request the frontend makes to those APIs must be tested. The frontend must also be able use the resources it receives in HTTP responses. This must also be tested. The user interface must also be tested so that it can be verified that the user interface meets its specification. This may be done manually. The last thing to test is application and business logic. For example, when adding projects to a playlist it should not add the project if the same project already is present in the playlist. And when removing a project from a playlist, the project should actually be removed from the playlist.

# Which code will not be tested and why?

The code that we didn’t write ourselves (library’s, generated code etc.) will not be tested. We won’t test this code because it is already tested by the .NET developers themselves. it is not useful and necessary to test this again.

# What type of testing is used and why?

The types of testing we are using are Unit and Integration testing. We are using both these types because we want to be able to test every bit of code that has functionality. We are using Unit tests for testing small units in the backend and frontend and we use Integration tests to test if the integration between backend, frontend, API and database works.

# Which test tooling will be used

**Unit:**

Backend: Xunit

Frontend: Jest

**Integration:**

**Backend:** Xunit

**Frontend:** Jest

**End to end:**

**Frontend:** Jest