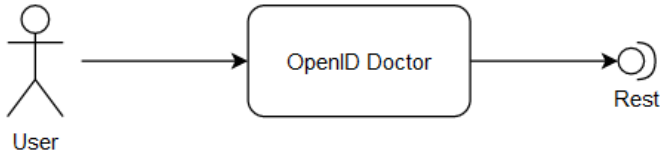
Software Architecture

OpenID Connect Doctor

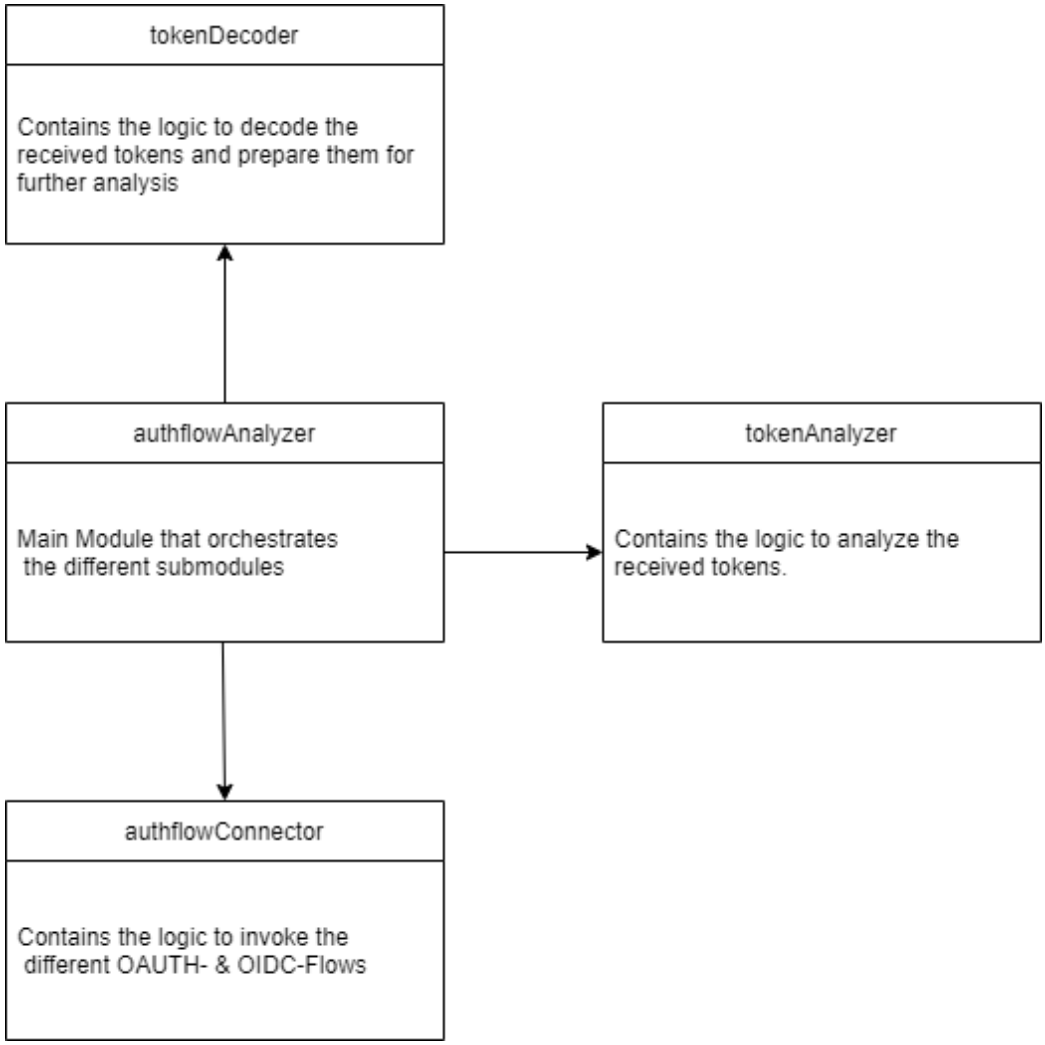
Project 8

1 Runtime Components



The OpenID Connect Doctor is a standalone CLI tool, that should be able to run on as many Operating Systems as possible. Furthermore, the application is required to work without the user having to install any additional programs. For this reason, it was decided, that the OpenID Connect Doctor should consist of only a single NodeJS application and use as few dependencies as possible. As the OpenID Connect Doctor must request tokens from an authorization server using OpenID Connect, it requires an interface to connect to a given authorization server using REST. To fully fulfill its task, the OpenID Connect Doctor is split into multiple smaller internal components, which carry out different tasks like requesting a token, decoding a token, etc.

2 Code Components



The code of the application will consist of multiple smaller classes, that each are responsible for a certain part of the application:

authflowConnector : This class will contain the logic to send HTTP-requests to the different Identity-Providers and process the returned HTTP-response (e.g. for the different OIDC-Login-Flows).

tokenDecoder : This class will contain the logic to decode and transform the received access and id-tokens into a format that can be analyzed by the rest of the application.

tokenAnalyzer : This class will contain the logic to analyze the prepared information and report these information to the user (e.g. via the terminal or a logfile)

authflowAnalyzer : This class will reference the other three class and will be responsible for orchestrating the different functions. Additionally it will also include input-validation, global error-handling and configuration.

The reason for this split is to keep a certain level of separation between the different parts of the software, so it is easier to modify and test them.

3 Technology Stack

NodeJS

In our project we are using Node Js as our programming language. Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.Node.js represents a "JavaScript everywhere" paradigm,[6] unifying web-application development around a single programming language, rather than different languages for server-side and client-side scripts.

- License: MIT License
- Operating system: z/OS, Linux, macOS, Microsoft Windows, SmartOS, FreeBSD, OpenBSD, IBM AIX
- Type: Runtime environment
- Version: 16.15.0

NPM

With Node Js as programming language, we are using npm is a package manager.It's a JavaScript programming language maintained by npm, Inc.

npm is the default package manager for the JavaScript runtime environment Node.js. npm can manage packages that are local dependencies of a particular project, as well as globally-installed JavaScript tools.

- License: Artistic License 2.
- Platform: Cross-platform
- Type: Package manager
- Version: 8.5.5

GIT

For the version control, we are using git in our project. Git is a software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development.

- License: GPL-2.0-only
- Operating system: POSIX (Linux, macOS, Solaris, AIX), Windows
- Type: Version contro
- Version: 2.36.1

The customer has suggested nodejs as the preferred language, because it has been used very much in their own technology stack. nodejs 16 is an LTS version with long-term support. Additionally it is available for Windows and openSUSE Tumbleweed (Linux). That is the reason for this suggestion, that the software can be developed based on multiple operating systems.
The package manager nmp 8.5.5 is integrated into nodejs 16. Therefore, we are using this version.