# DSEC: A Data Analyzer tool for ensuring secure software development life-cycle

## SCORE Report

2023

**Team members**
Giovanni Demasi- *PoliMi*
Lorenzo Poletti - *PoliMi*
André Schjøth - *PoliMi*
Kristian Djaković - *FER*
Lucija Strejček - *FER*
Jan Roček - *FER*

**Supervisors**
Raffaela Mirandola - *PoliMi*
Igor Čavrak - *FER*

**Sponsors**
Vikrant Kaulgud - *Accenture Labs*
Kapil Singi - *Accenture Labs*

# 1 Introduction

This document aims to outline the implementation process of the DSEC application, both from a team management point of view and a technical one.
The project has been developed as part of a Distributed Software Development course both from students of the University Polytechnic of Milan (Italy) and the University of Zagreb (Croatia).
The goals of the application and requirements (functional and non-functional) will be discussed, explaining the architectural design used, the implementation, and the testing process.

## 1.1 Background

SSDLC stands for Secure Software Development Life Cycle. The only realistic way to build secure software is to build security into each step of the cycle from inception to release – and this is how to create a secure SDLC (SSDLC).
Building a secure SDLC can be seen as adding a layer of security insulation to the bare SDLC pipeline. Each stage of the SDLC has corresponding security aspects that need to be considered directly in the tools and workflows used in that phase.

The DSEC service aims to verify that these layers of security insulation have been correctly added.

The commissioned agent of the project is ”Accenture Lab”, an Innovation Hub that partners with businesses in their innovation journeys through disruptive models and advances in technology, each of these elements aligns extremely well with their business challenges and industry context, enabling them to pivot to the New.

## 1.2 Goals

The main goal of this project is to develop a web service that can be used by anyone interested in building secure software, so by anyone who follows an SDLC that is easily accessible and easy to use, even without having enough experience in correctly implementing an SSDLC. The system should interface with the tools used to verify security compliance in an easy way for the user.

The main goal of DSEC is to develop a web service that allows Software Engineers to develop an SSDLC according to their needs. Implementing a correct SSDLC is a complex task and requires expertise in different fields, such as cybersecurity and policies governing data usage. Therefore, DSEC wants to gather all the information under one domain, so it is easier for users to understand what to do, by providing general compliance guidelines.

DSEC wants to primarily meet the needs of professional users. It can be used to help developers to find security hotspots and inconsistencies in different SDLC artifacts to suit the policies governing data.

## 1.3 Team Overview

Our team is made up of three students from the Polytechnic University of Milan (Italy) and three students from the University of Zagreb (Croatia). The distributed context was new to most of us and it required some initial set-up and a continuous effort to keep coordination and productivity high. To help with that, we chose to adopt the Scrum framework for our project to be able to improve our process from sprint to sprint.

| **Polytechnic University of Milan** | **University of Zagreb** |
|---|---|
| Giovanni Demasi | Kristian Djaković |
| Lorenzo Poletti | Lucija Strejček |
| André Schjøth | Jan Roček |

Even though we are all Computer Science and Engineering students this project still represented a challenge for everyone: the distributed environment, the Scrum approach, and the SSDLC brought us many new challenges and learning opportunities we had never faced before.

# 2 Requirements

## 2.1 Gathering Process

As Computer Science students, all the team members did not have so much experience in creating software from the beginning until the release, assessing all the potential security issues. Initially, the domain of the problem was not well known to us, which made the analysis of the problem harder.

Firstly, each team member individually analyzed the project assignment to try to understand all the requirements, trying to understand the needs of potential users. In a dedicated meeting, it has been possible to extract mandatory requirements and some lower-priority features.

It has been chosen to brainstorm ideas, let everyone propose their own, and annotate them. In this way, every team member was actively involved, which was crucial especially in the early stages of the project when it was still necessary to know each other. The next step was to compare the ideas and merge analogous ones, evaluating their feasibility and setting a level of priority.
Having a meeting with sponsors helped us in clarifying some aspects that were not initially clear and it helped us in refining our requirements.

## 2.2 Actors Involved

To have a clear description of requirements, the different actors in the system should be clearly defined.

| Actor | Description | Parent |
|---|---|---|
| Visitor | Everyone that can interact with the system (anyone that connects to the DSEC web service). The visitor is unauthenticated on the DSEC website. | None |
| User | Every person that is authenticated on the DSEC website. | Visitor |
| Moderator | A User, part of the development staff, who is granted the highest privilege level to adjust and change the content of the website. | User |

## 2.3 Functional Requirements

Some of the most important functional requirements derive directly from the goals of the project. In particular, they describe the functions needed in the system to achieve the desired goal, which is the possibility for users to analyze a code repository and examine the analysis result. In the following table, these requirements will be described, and it will be specified their level of priority, from high to low.

| Id | Name | Description | Priority |
|---|---|---|---|
| FR1 | Register | The system shall allow a Visitor to register an account in the system, providing an email, a password, and profile details | High |
| FR2 | Login | The system shall allow the user to log in with their credentials. | High |
| FR3 | Account info editing | The system shall allow the user to edit his password and his account details | Medium |
| FR4 | Account deletion | The system shall allow the user to delete and deactivate his account, deleting all his data from the system. | Low |
| FR5 | The user explores SSDLC | The system shall provide the user access to an explore page that explains general best practices to correctly implement an SSDLC. | High |
| FR6 | New analysis | The system shall allow the user to request a new analysis using the analysis web page. | High |
| FR7 | Repository linking | The system shall allow the user to link the GitHub project repository to perform a new analysis on the analysis page. | High |

| | | | |
|---|---|---|---|
| FR8 | The user selects the type of project | The system shall allow the user to select the type of project to be analyzed. | High |
| FR9 | User expresses project analysis preferences | The system shall allow the user to select which analysis aspects are more important for the project to analyze. | High |
| FR10 | The user starts a new analysis | The system shall allow the user to start a new analysis after having provided the requested information. | High |
| FR11 | System notifies user | The system shall allow the user to know the advancements in the analysis progress. | Medium |
| FR12 | The user aborts analysis | The system shall allow the user to abort an analysis that is in process, at any time during the process. | Low |
| FR13 | The user examines the analysis result | The system shall allow the user to visit a project page and examine, through a checklist, all his project analysis results. | High |
| FR14 | The user downloads the analysis report | The system shall allow the user to download a PDF report containing the analysis result. | Medium |
| FR15 | SSDLC guidelines update notifications | The system shall notify the user when one of his analyzed projects is affected by a change in SSDLC best practices. | Low |
| FR16 | The user manages the analysis data | The system shall allow the user to delete/archive analysis results of past project analysis, deleting all information from the system. | Low |
| FR17 | Guidelines/tools priority matrix | The system shall manage the user project type and preferences to rank the guidelines, from high to low priorities. | High |
| FR18 | Analysis tools | The system shall perform, using integrated analysis tools, all the necessary analyses to verify compliance with the guidelines with the highest priority. | High |
| FR19 | Moderator manages website content | The system shall allow the moderator to easily update the guidelines and the content of the website. | Medium |
| FR20 | Chatbot | The system shall allow the user to use a chatbot to interact with website moderators/cybersecurity experts to receive assistance/suggestions. | Low |

## 2.4 Non-Functional Requirements

| Id | Name | Description |
|---|---|---|
| NFR1 | Extensibility | The DSEC software must be designed in such a way that allows the easy implementation of extensions. |
| NFR2 | Security (user space) | The system should implement proper Authorization, so no one edits data related to other users. Only owners shall be able to view and edit their own project information. |
| NFR3 | Security (user data) | User data such as login credentials must be transferred using current state-of-the-art cryptography techniques. |
| NFR4 | Scalability | The system must tolerate (as in avoiding a significant performance loss) a sudden change in the number of online users. |
| NFR5 | Maintainability | The system must be designed in a way that facilitates the maintenance of the system code. |
| NFR6 | Performance - generic response time | The system must be in line with commercial SaaS systems already available for what concerns the web interface response time. |
| NFR7 | Accessibility | The website should be implemented according to WCAG 2.1. |

# 3 Project Management

## 3.1 Development Process

### 3.1.1 Roles

The team decided to implement SCRUM using static roles and sprints. The choice of the Scrum Master was made during the second internal meeting and given his experience in the Scrum process, the team choice was unanimous and led up to Kristian Djaković.

Instead, the product owner role has been assigned to André Schjøth. This decision didn’t come with particular criteria, as in the case of the SCRUM master, because nobody had previous experience. We asked internally who was interested and André volunteered.

### 3.1.2 Sprints

The project lasted 14 weeks, from the 10th of October 2022 to the 9th of January 2023. As decided by our project supervisors the sprint duration was of two weeks. Each sprint was used to address a different stage of the project, here is a summary:

| Sprint # | Start date | End data | Stage |
|---|---|---|---|
| 0 | 2022-10-11 | 2022-10-17 | Intro/Vision/Plan |
| 1 | 2022-10-18 | 2022-10-31 | Requirements |
| 2 | 2022-11-01 | 2022-11-14 | Initial Development |
| 3 | 2022-11-15 | 2022-11-28 | Alpha Prototype |
| 4 | 2022-11-29 | 2022-12-12 | Beta Prototype |
| 5 | 2022-12-13 | 2022-12-26 | Refinements/Documentation |
| 6 | 2022-12-27 | 2023-01-09 | Final Prototype/Documentation |

At the beginning of each sprint, we held a sprint planning meeting to define the goals of the sprint in terms of features to be implemented. In the end, we had the sprint retrospective and review, with all the team members, in which we actively discussed our achievements and problems we faced, collecting the outcome in a sprint report.
Moreover, almost all the weeks, we had a one-hour meeting with sponsors where we showed the advance in the product and discussed, by following their feedback, the following steps.

Daily Scrum meetings were not feasible due to the different team member’s academic schedules. We then created a “daily” Slack channel with a daily reminder in the morning in which each team member was able to update colleagues about “what he did" and “what he is going to do". Each team member took care of the task assigned to him, updating its status. At the end of the sprint, the unfinished tasks were delayed to the sprint after.

### 3.1.3 Product and Sprint Backlog

The product backlog is an ordered list of what is needed to improve the product. At the beginning of each sprint, a subset of items from the product backlog is selected based on the priority of the requirements. Items related to high-priority requirements are selected first. From the selected product backlog subset we create a list of detailed tasks, which describe what the frontend and backend teams should tackle. To split the work evenly, for each task we assign an estimation of story points, which represent the amount of time that should take to complete. Then, members select which tasks are more suitable for them, based on their knowledge and individual time constraints.
Since this could lead to an uneven task distribution, after this process the SCRUM master recalibrates the workload by re-assign tasks to members that have less work.

### 3.1.4 Git Branching Strategy

We used Git as our version control system and remotely hosted our repositories on GitHub. The codebase was composed of three repositories, one for the frontend of the application, one for the backend, and one for the integrated tools' scripts. We decided to use Git workflow as our branching strategy, this type of git workflow helped us keep track of the

various features developed at the same time and allowed us to efficiently coordinate our work.
Each feature and each fix was developed in its dedicated branch and when it was completed a pull request was opened, to merge the new code on the develop branch.
The developer that opened the pull request assigned reviewers to it, who needed to check the code and leave comments, suggest changes and ultimately approve the pull request.
We also made use of the "Issues" tool provided by GitHub: a developer could open an issue at any time on a repository, explaining briefly what they thought was a problem with the codebase or describing what feature he was going to implement.
Here is our GitHub organization.

## 3.2 Team Organisation

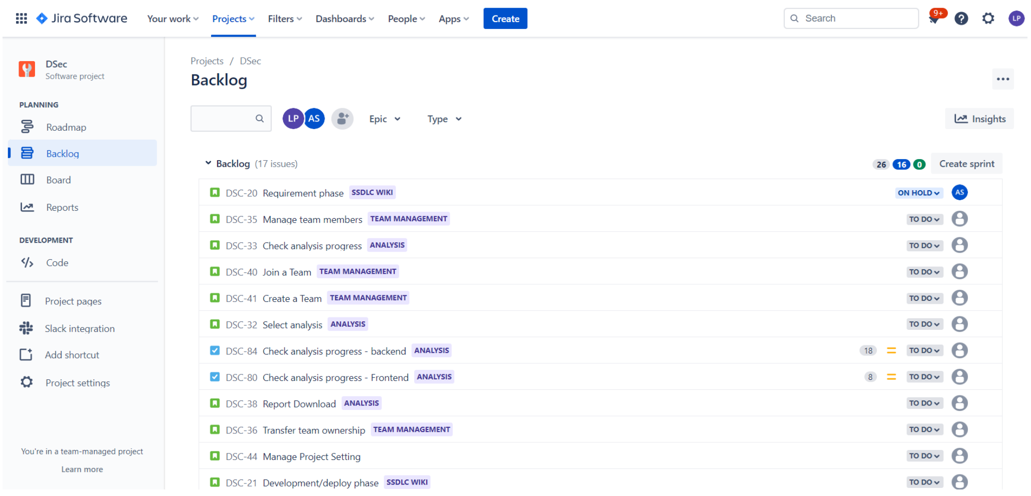
As will be shown in the architectural design section, the system is made of two macro-areas: frontend and backend. By a preliminary analysis of the project, it turned out that the number of tasks related to different macro-areas was pretty much the same, hence we decided to evenly split our team into two subteams and assign one area each.
In particular, the backend team was formed by Giovanni Demasi, Lorenzo Poletti, and Jan Roček, and, Kristian Djaković, Lucija Strejček, and André Schjøth were part of the frontend team. The subteams have been formed by taking into account the experience and expertise of each team member. There is a list of pros and cons that are followed up by such a choice.
*Pros*:
- Each member of the group needed to specialize only in a few technologies.
- There was no need of keeping every component of the team aware of technical peculiarities relative to the implementation of every macro-area.
- We noticed that communication significantly improved within each subteam. This is probably because the subteams were composed of three people instead of six, making communication more clear and direct.

*Cons*:
- The system integration required more time than expected.
- The communication between the two subteams has been penalized, creating potential misalignments between the frontend and the backend.
- The assessment of the testing procedures was non-trivial. As it will be better defined later, we ended up testing the frontend with manual inspection of corner cases and the backend with integration tests.

## 3.3 Team Coordination

In a distributed environment, team coordination is crucial for the project's success, therefore the first days of the first sprint were dedicated to the choice of the most suitable communication tools, according to the features provided and team members' experience.
The tools that the team has been using during the project are the following:
- GitHub: a standard tool for code versioning. Each subteam worked on its own repository, making deployment when needed.
- Jira: project organization tool. Our team decided to use this tool since one of the team members, the scrum master, has had previous experience in using it. We were able to keep a clean board, with an organized backlog and sprints.

- Slack: main communication tool. Different textual channels for teams and sub-teams were created to manage different topics and to facilitate teamwork.
- Google Drive: shared folder and online collaboration on documents. To keep the content organized and clear we created subfolders for Minutes-Of-Meetings, Archive, Documentation, and Diagrams.
- Google Meet: communication tool used for internal meetings and meetings with supervisors through a recurrently scheduled room. It represents a good choice since it is free and can be used directly from the browser and allow multiple screen sharing at the same time.

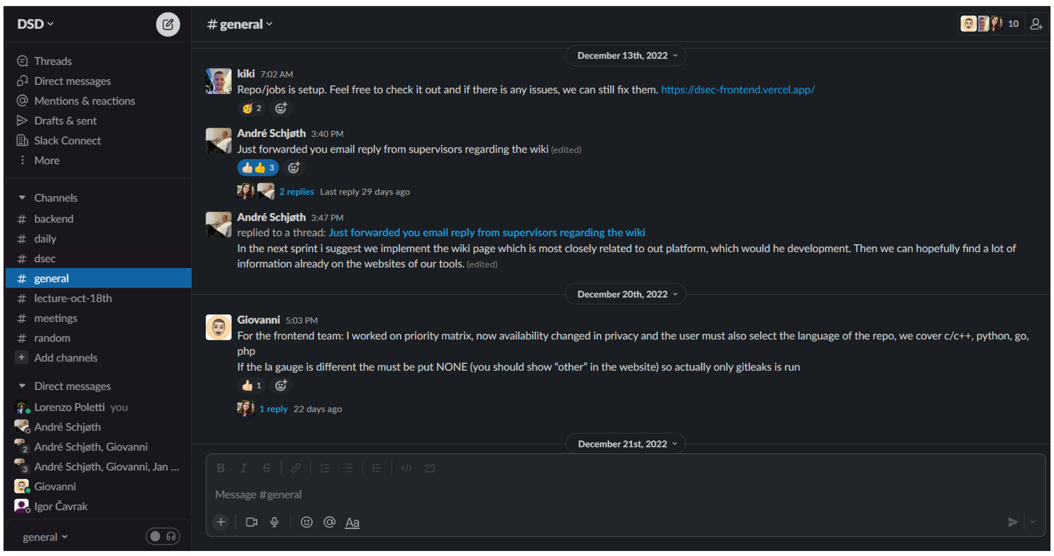


Backlog

Backlog (17 issues) 26 16 0

| | | | | | |
|---|---|---|---|---|---|
| DSC-20 | Requirement phase | SSDLC WIKI | | ON HOLD | AS |
| DSC-35 | Manage team members | TEAM MANAGEMENT | | TO DO | |
| DSC-33 | Check analysis progress | ANALYSIS | | TO DO | |
| DSC-40 | Join a Team | TEAM MANAGEMENT | | TO DO | |
| DSC-41 | Create a Team | TEAM MANAGEMENT | | TO DO | |
| DSC-32 | Select analysis | ANALYSIS | | TO DO | |
| DSC-84 | Check analysis progress - backend | ANALYSIS | 18 | TO DO | |
| DSC-80 | Check analysis progress - Frontend | ANALYSIS | 8 | TO DO | |
| DSC-38 | Report Download | ANALYSIS | | TO DO | |
| DSC-36 | Transfer team ownership | TEAM MANAGEMENT | | TO DO | |
| DSC-44 | Manage Project Setting | | | TO DO | |
| DSC-21 | Development/deploy phase | SSDLC WIKI | | TO DO | |

*1 - DSD Jira Backlog page*



# general

December 13th, 2022

**kiki** 7:02 AM
Repo/jobs is setup. Feel free to check it out and if there is any issues, we can still fix them. https://dsec-frontend.vercel.app/
2

**André Schjøth** 3:40 PM
Just forwarded you email reply from supervisors regarding the wiki (edited)
3
2 replies Last reply 29 days ago

**André Schjøth** 3:47 PM
replied to a thread: Just forwarded you email reply from supervisors regarding the wiki
In the next sprint i suggest we implement the wiki page which is most closely related to out platform, which would he development. Then we can hopefully find a lot of information already on the websites of our tools. (edited)

December 20th, 2022

**Giovanni** 5:03 PM
For the frontend team: I worked on priority matrix, now availability changed in privacy and the user must also select the language of the repo, we cover c/c++, python, go, php
If the la gauge is different the must be put NONE (you should show "other" in the website) so actually only gitleaks is run
1
1 reply 22 days ago

December 21st, 2022

*2 - DSD Slack general channel*

## 3.4 Risk Management

Risks are part of everything, so we made a list of possible risks and found a way to try to keep them under our control.
- Underestimate the task's difficulty, coming at the end of the sprint with unfinished tasks.
- Poor communication.

- Unbalanced workload between team members.

While in theory, the countermeasures to these risks are quite clear, in practice it becomes challenging to establish good practices. This is due both to the number of people involved who have different backgrounds and cultures. Moreover, nobody has strong experience in managing such a project, so the following countermeasures are the result of months of trial and error.

- Each team member informed the team regarding his own personal and academic engagements, such as travels and mid-term exams.
- Find a good estimated time for each sprint's task, trying to be realistic and evenly splitting tasks among the team members.
- Using the daily Slack channel and weekly meetings to keep up with what each team member was doing, being ready to rebalance the split work if needed.

To give a concrete example, during the third sprint, we encountered some unbalanced workload between team members, in particular with the backend team. This was due to different academic backgrounds and poor communication. So we tackled the problem during the sprint retrospective and decided to better assign the tasks during the sprint planning, ensuring that all the team members had the same amount of workload.

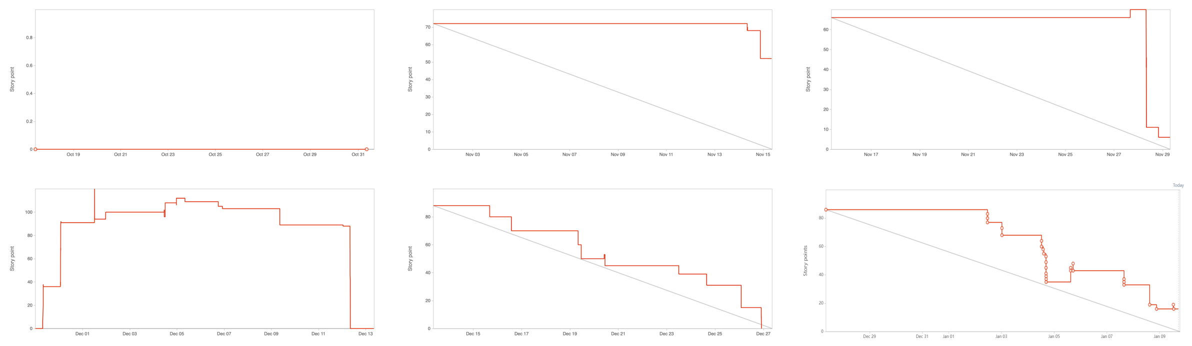
## 3.5 Project Metrics

### 3.5.1 Sprints Backlogs

At each sprint, we created a backlog taking some tasks from the whole backlog created at the beginning of the project, based on the functional requirements. Here is the last backlog from our Jira board.

### 3.5.2 Sprints Burndown Charts

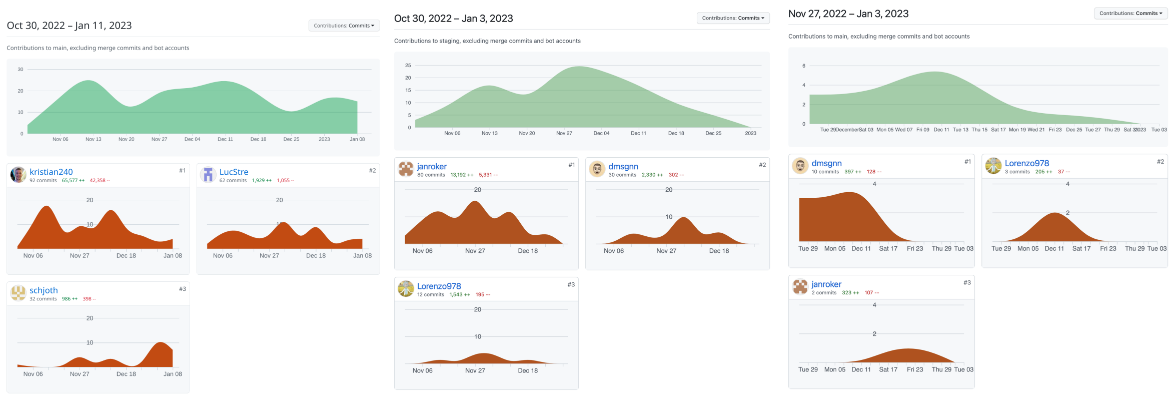
All the burndown charts of each sprint are shown below, except the one of sprint 0 since it was dedicated to better knowing the group. Initially, we had some trouble making a nice burndown chart because we closed all the tickets only at the end of the sprint.
Overall, we are satisfied since we improved by the end of the project and this is part of the growth and learning process.



*3 - Sprint burndown charts, from Sprint 1 to Sprint 6*

### 3.5.3 GitHub Metrics

Here are reported the GitHub repositories metrics of team members, respectively for the frontend, backend, and tool scripts repositories.



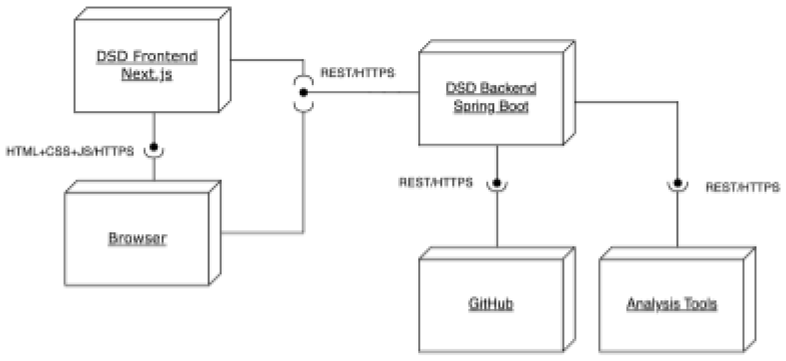
*4 - GitHub metrics of Frontend, Backend and Tool Scripts repositories respectively*

For the backend repository, we had a variable number of active branches depending on the features needed. At least two branches (staging and develop) were always present, then for each feature or bug to solve new branches were created for then being merged into staging.
The total number of pull requests is 23 and follows the time distribution of individual commits because pull requests were created straight after commits were pushed in the feature development branch.

The tool-scripts repository instead has fewer commits (19) and pull requests (3), since we used it only to create docker files for the tools. The contributions for each member are different from the backend, but still the commits and pull requests distribution is centered around the alpha and beta version sprint.

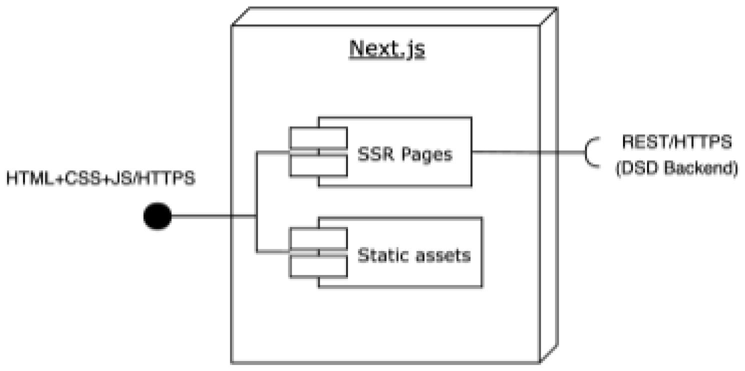
For the frontend repository, we had two branches (main and develop) that were always present and additional branches were created for each feature or fix. After all commits were pushed to the new branch we would create a pull request. The total number of created and closed pull requests is 26. After the pull request was approved by other frontend team members it was merged into the develop branch and the feature branch was deleted. At the end of each sprint, when all the required features and fixes have been merged into the develop branch, the main branch is updated by merging with the develop branch, and the code is deployed.
Each member contributed, but with differences in the number of lines. This is mostly due to different skill sets and task assignments. For instance, some members had to learn a new programming environment and others spent more time on the design. The majority of commits have been done during the alpha and beta version sprints.

## 4 Architectural Design



**High-level architecture**

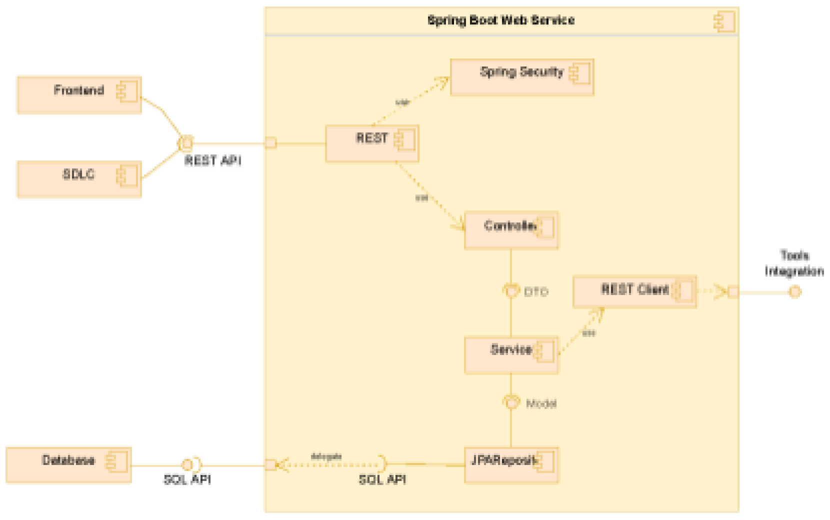
*5 - High level architecture of DSEC system*



**Frontend architecture**

*6 - Frontend architecture of DSEC system*

The frontend architecture is based on the Next.js (React) framework.



**Backend architecture**

*7 - Backend architecture of DSEC system*

The backend web service exposes a REST API that follows the REST architectural style, which is an architectural style that uses a request-response communication pattern.

In REST, the client sends a request to the server, and the server sends back a response. The request and response are typically in the form of HTTP messages, and the communication between the client and the server is stateless, meaning that the server does not store any information about the client between requests.

The REST API is made up of a set of endpoints, which are URLs that correspond to specific actions or resources. Each endpoint is associated with a specific HTTP method (e.g., GET, POST, PUT, DELETE) that specifies the type of action being performed on the resource.

Backend web service has a typical Spring Boot REST service architecture and is made of the following parts:

- **Repository**: The repository is a component that is responsible for storing and retrieving the model objects from a database.
- **Model:** The model or entity is a Java object that models the data that is stored inside a database.
- **Service layer**: The service layer is a component that encapsulates the business logic of the application.
- **REST controller:** The REST controller is a component that exposes the service layer as a RESTful API.
- **DTO**: The DTO is a Java object that represents the data being sent to or received from the server. In a Spring Boot REST service, the model is typically defined as a POJO (Plain Old Java Object) with some fields and getter/setter methods.

# 5 Implementation

## 5.1 Frontend

The technology used on the frontend is Next.js (React) framework. Next.js has a file-system-based router built on the concept of pages. When a file is added to the pages directory, it's automatically available as a route. The app exposes 10 pages and many more routes since there is one dynamic page. Defining routes by using predefined paths is not always enough for complex applications. To create a dynamic route, brackets need to be added to a page ([param]). Alongside these routes, there are some static assets (images, fonts, …) that are also available via the same interface.

All routes are rendered on the server side, which means that a user is getting a pre-build page. Once a document is requested, Next.js will pre-build a page in runtime and once the page response is built, it will return an HTML response.

When a page is loaded in a browser, the browser will paint the screen, and React will start the hydration process. This process is needed so React can come to the same status as it was on the server side. While the hydration process is not finished, the user won’t be able to do any interactions.

## 5.2 Backend

The team developing a web service is using Spring Boot and Python to build their system. The choice of Spring Boot for the backend API is based on the experience of team members and the advantages of the framework. Spring Boot makes it easy to create stand-alone, production-grade Spring-based applications with features such as an embedded web server, automatic configuration, and easy integration with other libraries and frameworks. Additionally, the application is configured using Spring's Dependency Injection (DI) framework, which allows for easy management and swapping of implementations in different environments. By using different profiles, the application can specify a configuration that is specific to a particular environment, such as the development, test, or production environment.

This allows for a smooth transition of the application from one environment to another. The application uses an embedded H2 database in development and test environments, while it uses a PostgreSQL database in the production environment. This allows for more efficient use of resources during development and testing while ensuring that the production environment has a robust and reliable database to support it.

The application also exposes RESTful APIs that allow clients to interact with the application over HTTP. These APIs are implemented using Spring's MVC (Model-View-Controller) framework, which separates the application's logic, data, and presentation layers. This allows for a clean separation of concerns and improves the maintainability of the application. Furthermore, the decision to use Python for the development of scripts that run and collect analysis tool results was made because it enables the team to quickly prototype and achieve a unified API for each tool that is integrated with the backend web service. Additionally, Python's large ecosystem of libraries and tools allows for rapid development and can save development time.

## 5.3 Integration

The distribution of work among subteams speeded up the development time of every feature but, on the other hand, increased the complexity of system integration. To prevent misunderstandings, possibly due to a lack of communication between subteams, we decided to adopt periodic updating policies on our communication channel and to use swagger to have API documentation.
The communication between the backend and frontend has been led by the backend subteam, this is because we thought it would be easier for the frontend team to adapt to our decision rather than the opposite.

# 6 Verification and Validation

This project has been tested both automatically and manually both during and after implementation.
Some internal services and modules, like backend APIs, were tested while the implementation was still going. On the other hand, some features like part of the frontend, need to be tested manually after the implementation of all the components on which they depend.
The analysis tools integrated with the DSEC web service have not been tested, each tool is a third-party service and it is assumed to work correctly. What has been tested is the integration of the tools and communication between the backend and python script which executes and exposes a tool.

## 6.1 Backend

Backend testing has been implemented using JUnit, one of the most popular unit-testing frameworks in the Java ecosystem. Since the backend server is mostly dedicated to the creation of APIs, it was more suitable for traditional integration testing since more than one module at a time was tested to assure the correct behavior of the API.
There was no need for mocking components as the tests are run in a containerized backend using Test Rest Template. This allowed us to also test sequences of actions that can be performed by a potential user and verify the absence of unexpected behaviours. In particular, the database always starts as a new and fresh instance with no data in it, an initial setup script creates all the databases and the design documents associated with them, and then another script sets up the needed environment and creates the entities used in the tests.
Some examples of tests that have been performed aimed to verify that the login was successful after registration or that a registration returned a *bad request* response if email or password constraints were not respected.

## 6.2 Frontend

Frontend testing has been implemented using jest.js, one of the most popular unit-testing frameworks in the JavaScript ecosystem.
Some utility hooks and components needed to be mocked, those are mostly connected to routing and APIs connection. The mocking is implemented using the default mock methods provided by jest. Also, all components have been tested with snapshots as well.

## 6.3 Manual Testing

A lot of manual testing has been performed during the entire project development. It has been done in parallel with implementation, simultaneously to automatic testing, to verify that the behavior of the developing feature was the desired one.
For instance, backend manual testing was performed using Postman to send requests to the server in combination with the H2 database console, to verify that entities were correctly added to tables and that the response body and type were correct.

Manual testing has also been intensively used to verify the correctness of the communication between the backend server and the python scripts of the tools.
Other manual tests have been described in detail with related procedures and acceptance criteria in the acceptance test plan, to make the validation of the final product. Those tests can be reproduced directly by the end-user on the deployed website.

## 6.4 Non-Functional Requirements

### 6.4.1 Extensibility and Maintainability

Extensibility and maintainability have not been explicitly tested, they have been taken into account during the entire development process using clean code strategies (short methods, enumerations, not hard-coded information, etc…).
In addition, some automatic tools have been used like Sonar Cloud, to avoid the presence of duplicated code and code smells. In addition, adding a new analysis tool to the system does not require code changes, but only script and endpoint creation.
Also, a CMS has been used to make the website content easily changeable, without changing the frontend code.

### 6.4.2 Security: User Space and User Data

Security measures have been taken into consideration during implementation, saving passwords and tokens in an encrypted way and making personal information accessible only to the respective owner. Whenever a token is sent, it is encrypted by the sender and decrypted by the receiver using trusted encryption-decryption protocols. Encryption-decryption and restricted accesses (unauthorized error response) have been tested both with manual and automated testing.

### 6.4.3 Performance and Scalability

Performance and scalability have been automatically tested using the Apache Jmeter application, an open-source software designed to load test functional behavior and measure performance. Most of the endpoints created have been tested both for performance and scalability, using single performance tests or structured stress tests in which a lot of users do the same action in a very short period of time.

### 6.4.4 Accessibility

Accessibility testing has been automatically done on pages using an open-source tool called axe-core and WAVE. All pages should be at least level A compliant. Most of the pages, user flows, and components have been tested.

# 7 Outcomes

During these months, the team has developed and deployed a platform of tools for analyzing the user's software. The tool contains support for multiple programming languages and has also included some general tools such as GitLeaks, which is not specific to certain programming languages. This approach has resulted in a platform that can be used in many different projects. As certain projects might already have pipelines that cover the same phases of SDLC as the platform, the platform might not be as useful for everybody. However, the addition of the wiki makes the website useful for more projects, as it also covers other parts of the SDLC.

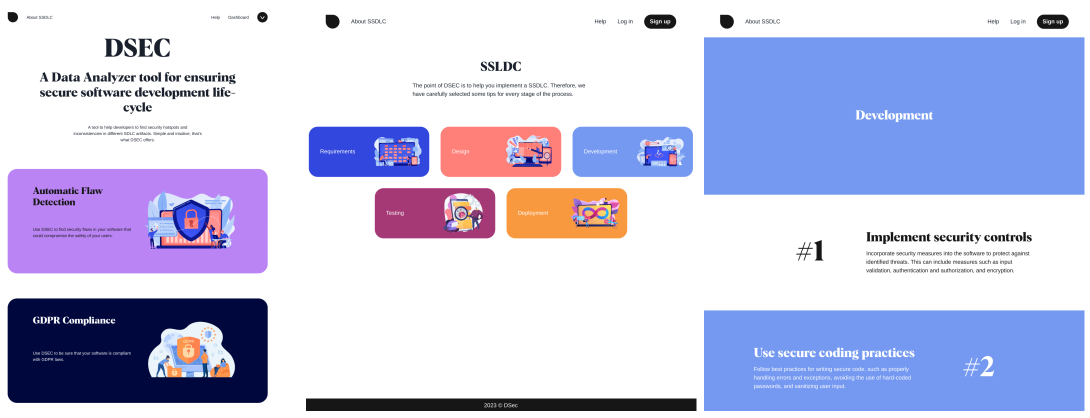
During the project, the communication with supervisors and customers resulted in a prioritization of the platform rather than the wiki. Some of the reasoning behind this was the limited technical complexity of the wiki, which would not provide the students with the desired technical learning outcome of the respective course. The prioritization of the platform has resulted in the platform including more functionality than the requirements specified, and the wiki not fulfilling all its requirements.

## 7.1 Wiki

In the wiki, the team has gathered some tips for every phase of the SDLC. These tips are generalized to target a larger audience. The amount of content could definitely be improved. However, the implementation is easily extendable which would make future addition easy to achieve. As specified in the requirements, the goal was to have support for a moderator user, but this was one of the requirements that were down-prioritized from the original prioritization due to the aforementioned dialogue with supervisors and customers.

## 7.2 Platform

As mentioned earlier the team implemented more features in the platform than originally specified, such as supporting several languages: C/C++, Go, Python, and PHP.



DSEC

A Data Analyzer tool for ensuring secure software development life-cycle

A tool to help developers to find security hotspots and inconsistencies in different SDLC artifacts. Simple and intuitive, that's what DSEC offers.

Automatic Flaw Detection

Use DSEC to find security flaws in your software that could compromise the safety of your users.

GDPR Compliance

Use DSEC to be sure that your software is compliant with GDPR laws.

SSLDC

The point of DSEC is to help you implement a SSDLC. Therefore, we have carefully selected some tips for every stage of the process.

2023 © DSec

Development

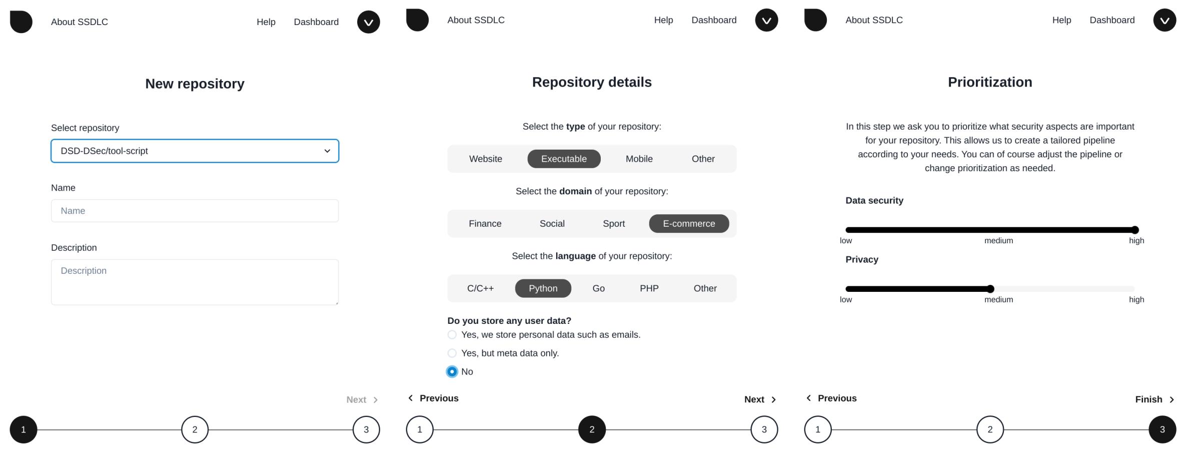
Implement security controls

Incorporate security measures into the software to protect against identified threats. This can include measures such as input validation, authentication and authorization, and encryption.

Use secure coding practices

Follow best practices for writing secure code, such as properly handling errors and exceptions, avoiding the use of hard-coded passwords, and sanitizing user input.

*8 - Homepage, Wiki overview, and a wiki page with SSDLC tips.*


About SSDLC  Help  Dashboard

**New repository**

Select repository: DSD-DSec/tool-script

Name: Name

Description: Description

Next ›

About SSDLC  Help  Dashboard

**Repository details**

Select the **type** of your repository: Website | [✓ Executable] | Mobile | Other

Select the **domain** of your repository: Finance | Social | Sport | [✓ E-commerce]

Select the **language** of your repository: C/C++ | [✓ Python] | Go | PHP | Other

**Do you store any user data?**
- Yes, we store personal data such as emails.
- Yes, but meta data only.
- [✓ No]

‹ Previous  Next ›

About SSDLC  Help  Dashboard

**Prioritization**

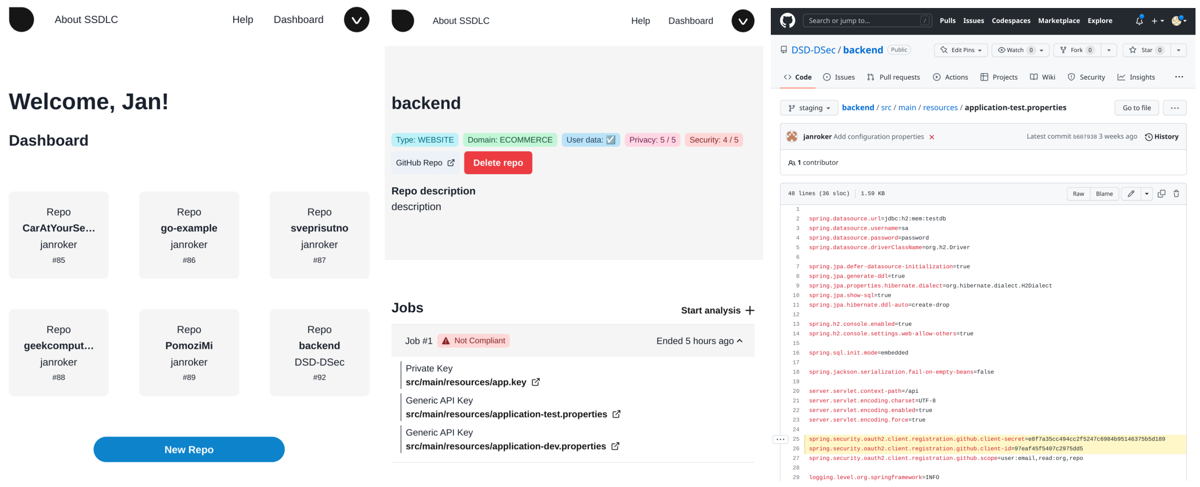
In this step we ask you to prioritize what security aspects are important for your repository. This allows us to create a tailored pipeline according to your needs. You can of course adjust the pipeline or change prioritization as needed.

**Data security**: low — medium — [✓ high]

**Privacy**: low — [✓ medium] — high

‹ Previous  Finish ›

9 - Adding a new repository in 3 steps: Link repository, provide details, and prioritize.


About SSDLC  Help  Dashboard

**Welcome, Jan!**

**Dashboard**

| | | |
|---|---|---|
| Repo **CarAtYourSe...** janroker #85 | Repo **go-example** janroker #86 | Repo **sveprisutno** janroker #87 |
| Repo **geekcomput...** janroker #88 | Repo **PomoziMi** janroker #89 | Repo **backend** DSD-DSec #92 |

New Repo

About SSDLC  Help  Dashboard

**backend**

Type: WEBSITE | Domain: ECOMMERCE | User data: ✓ | Privacy: 5 / 5 | Security: 4 / 5

GitHub Repo  Delete repo

**Repo description**
description

**Jobs**  Start analysis +

Job #1  Not Compliant  Ended 5 hours ago

Private Key
**src/main/resources/app.key**

Generic API Key
**src/main/resources/application-test.properties**

Generic API Key
**src/main/resources/application-dev.properties**

DSD-DSec / **backend** Public

staging  backend / src / main / resources / **application-test.properties**

**janroker** Add configuration properties  Latest commit b607938 3 weeks ago  History

1 contributor

48 lines (36 sloc) | 1.59 KB

```

spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.driverClassName=org.h2.Driver

spring.jpa.defer-datasource-initialization=true
spring.jpa.generate-ddl=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=create-drop

spring.h2.console.enabled=true
spring.h2.console.settings.web-allow-others=true

spring.sql.init.mode=embedded

spring.jackson.serialization.fail-on-empty-beans=false

server.servlet.context-path=/api
server.servlet.encoding.charset=UTF-8
server.servlet.encoding.enabled=true
server.servlet.encoding.force=true

spring.security.oauth2.client.registration.github.client-secret=e8f7a35cc494cc2f5247c6984b95146375b5d189
spring.security.oauth2.client.registration.github.client-id=97eaf45f5407c2975dd5
spring.security.oauth2.client.registration.github.scope=user:email,read:org,repo

logging.level.org.springframework=INFO
```

10 - DSEC platform with dashboard, results page, and link to GitHub files containing security flaws.

# 8 Future

A future implementation of the project would be the creation of a team to allow users to see the same analysis result of the team project. This feature was listed in the initial requirements but has not been implemented due to time constraints.
The concept of the DSEC service is innovative because it allows the end user to have a full understanding of the SSDLC compliance of his projects, in addition, it has a very high potential in the near future because more and more sensitive data are being produced and collected and privacy and security are topics of current interest and discussion. It is currently limited by the number of available analysis tools, which is something that could be quite easily overcome. To ensure that our service fully provides its abilities, we would need analysis tools for both requirements and test phases. Once provided coverage for most of the development life cycle phases, software engineers will be able to use DSEC service with only benefits, regarding additional guarantees about software security and user privacy, everything in one place.

# 9 Conclusion

At the time of writing, the DSEC is an available working web service that allows its users to analyze their GitHub project repositories to find security flaws. Right now, as agreed with project sponsors, due to our project time constraints, the available analysis tools cover only the build/deploy phase.
For reference, using our DSEC web service we were able to analyze the repository used for the development of this project to be aware of security flaws.

We can conclude that considering the full requirements list, we have successfully implemented 100% of High priority requirements and 75% of Medium priority requirements. Low-priority requirements were created as nice-to-have features, but due to time constraints, only a few of them (40%) have been implemented.

Given that the distributed environment and the Scrum approach were new to most of the members of the team, we are quite satisfied with how the work turned out, and we are sure that we will consider the Scrum framework again in the future for any project that requires an agile approach.

Last but not least, this project gave us the opportunity of teaming up with students with different cultures, skills, and perspectives, and even though sometimes we had some differences we are really glad we embraced this journey all together.

# 10 References

1. GDPR guidelines, https://gdpr-info.eu
2. DSEC web service, https://dsec-frontend.vercel.app/
3. DSEC GitHub organization, https://github.com/DSD-DSec

# 11 Demo

## 11.1 Source Code

The whole source code of the DSEC web service is splitted among four different repositories, all created under a common public GitHub organization called “DSD-Dsec”, used to manage the whole project. The main project repositories are three, a backend repository to manage the backend side of the project, a frontend repository to manage the frontend side of the project and a tool-script repository which contains the Python scripts used to run and manage the integrated analysis tools. An additional repository has been created containing files needed to run the project locally.

All repositories are accessible through the GitHub organization page, but, for completeness purposes, all the necessary links to fully access the source code are reported here:

1. “DSD-Dsec” organization, https://github.com/DSD-DSec
2. Backend repository, https://github.com/DSD-DSec/backend
3. Frontend repository, https://github.com/DSD-DSec/frontend
4. Tool scripts repository, https://github.com/DSD-DSec/tool-script
5. Local run repository, https://github.com/DSD-DSec/.github

## 11.2 Web Service demo

### 11.2.1 Deployed web service demo

The DSEC web service has been deployed and can be fully tested at the following link: https://dsec-frontend.vercel.app/. To have the best experience and avoid compatibility problems, it is suggested to test it through Chrome web browser.

### 11.2.2 Local web service demo

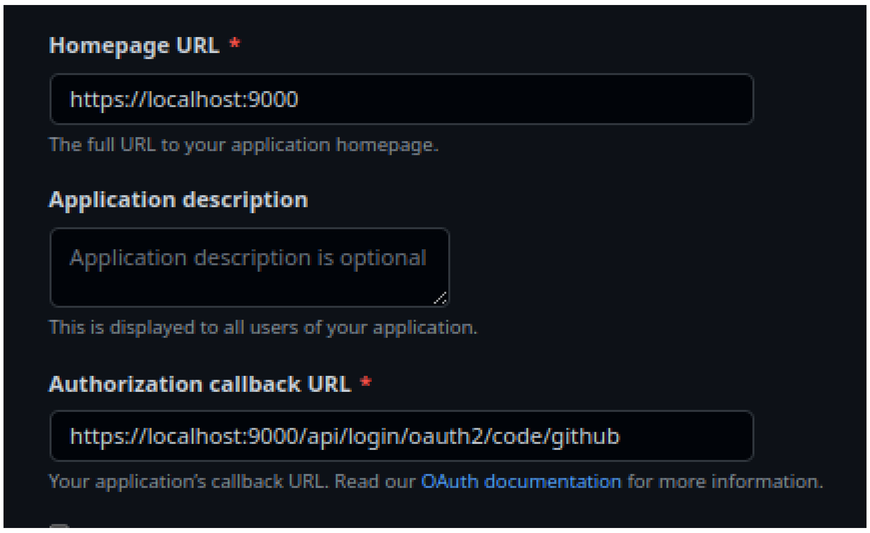
The DSEC web service can also be tested locally with some prerequisites. In particular, you will need to have:

1. ngrok or similar tool that is able to make your application accessible from an external network. This is necessary for github to be able to access the webhook on the backend.
2. docker and docker-compose.

In order to run the application, the following steps must be followed:

1. Clone frontend, tool-script, backend and .github repositories in the same folder;
2. Copy *docker-compose.yml* and *.env* files from the .github repository in the directory that contains the above mentioned repositories. The filesystem tree structure should look like:
    - .
        - .github/
        - backend/
        - frontend/
        - tool-script/

        - .env
        - docker-compose.yml
3. Start the ngrok tunnel with the *ngrok http https://localhost:9000* command. The url specifies the backend url on the host machine.
4. Modify the *DSEC_BACKEND_URL_PUBLIC* variable in the *.env* file. The value should be the *url* obtained in the third (3) step in the *Forwarding url -> something* line.
5. Modify other necessary values. The default ones should work for some time. The plan is to discontinue the github oauth2 app secrets in the near future. It might be necessary to set up your own github oauth2 app and replace *DSEC_GITHUB_CLIENT_SECRET* and *DSEC_GITHUB_CLIENT_ID* variables. The settings for the github oauth2 app should be as follows:



Homepage URL *

https://localhost:9000

The full URL to your application homepage.

Application description

Application description is optional

This is displayed to all users of your application.

Authorization callback URL *

https://localhost:9000/api/login/oauth2/code/github

Your application's callback URL. Read our OAuth documentation for more information.

*11 - settings for the github oauth2 app*

6. If the application is to be deployed, it would be insecure to use the default variables:
7. *cd* into the folder that contains the repositories and the *docker-compose.yml* file. Run the *docker-compose up* command. The application should be accessible on the *http://localhost* url;
8. Type the following address in the browser: *https://localhost:9000/api* - or similar if you have changed the backend service config;
9. In the tab that was opened in eighth (8) step, go to advanced and click accept risk. This is necessary because our backend app uses self-signed certificates.
10. Go to *http://localhost* and fully test the DSEC application.

It must be mentioned that after modifying the *DSEC_NEXT_PUBLIC_API_BASE_PATH* or *DSEC_NEXT_PUBLIC_GITHUB_REDIRECT_URL* variables the frontend Docker image must be rebuilt. To achieve the wanted result you must first run *docker-compose rm* and then remove the frontend image using the *docker image rm *-frontend:latest* command. After that you must repeat the seventh (7) step.

If modifications are done only to the variables used by the backend (any other variable) it is only necessary to repeat the seventh (7) step. If you have created some repositories but restarted ngrok and the ngrok url changed, you will not be able to run the analysis on the already existing containers because previously created webhooks point to the wrong url. The repos will need to be recreated in the app.

Same instructions can be found in the README of the .github repository at the following link: https://github.com/DSD-DSec/.github/tree/main/profile.

## 11.3 How to use

A “how to use” section, aiming to explain how to use the DSEC web service and how to easily understand its main functionalities, is provided  in the “help” section (accessible through the website’s navigation bar) of the DSEC application website.