PrivTAP

Privacy-preserving Trigger-action IoT Platform

Basic Enna, Benedetti Michele, Radonjic Ivan
Mälardalen University, Västerås, Sweden

Panseri Leonardo, Santoro Emanuele, Sassi Alessandro, Vaccarini Lorenzo Maria
Politecnico di Milano, Milan, Italy

May 17, 2023

1 Introduction

PrivTAP is a project proposed by Kulani T. Mahadewa, a senior lecturer at the University of Moratuwa, located in Sri Lanka. [29] Some of the main objectives of this project are to provide control to the users over their data that is needed for rule execution, to prevent data leaks through unauthorized flows, and to allow third-party Online Service Providers (OSPs) to integrate with the platform, giving users the possibility to utilize their services to create rules. The team working on this project consists of 7 people, where 3 of the students are from Mälardalen University (MDU) from Sweden, and 4 from Politecnico di Milano (POLIMI) from Italy. The project was also supervised by Alessio Bucaioni from Mälardalen University, and Vincenzo Riccio from Politecnico di Milano.

PrivTAP is a trigger-action platform that preserves the privacy of users' sensitive data. It is developed as a web-based system that facilitates the integration of multiple third-party online services to enable the creation and management of automation rules. It starts from the base idea of IFTTT [16] which grants the users a connection to two different OSPs (Online Service Providers). This project improves upon the original idea by giving the end user more control over their data by allowing them to choose the privacy level they want to grant to a service. This helps to prevent privacy weaknesses that might occur on the IFTTT platform. [27]

Moreover, it allows users to create trigger-action rules through a simple application and to integrate third-party OSPs for automation. The platform allows online service providers to publish trigger events or actions, and a user can specify automation rules in the form of "if a trigger event happens, then perform an action", by using an intuitive application deployed in the Cloud.

The developed project presents a prototype platform that includes the aforementioned features. Functionalities related to the preservation of user privacy were treated with high-level priority with respect to the others that are kept to a medium level of both priority and time effort. The project also includes the development of two service API endpoints that we are using to demonstrate the platform usage.

This document will provide its readers with an introduction to what the project is about and will give an insight into the adopted software engineering process (namely SCRUM), architectural

design and implemented requirements, the technologies used for the implementation of the project, and finally, some conclusions and lessons learned from this project.
Since this project was developed as part of the "Distributed Software Development" course, its main challenge consisted in overcoming the diversified team composition without being able to meet in person, since team members are not only from different universities but also from different countries. This challenge allowed the team members to gain valuable experience that will be beneficial for future projects and for working in a team.

1.1 Document Organization

The presented document is organized as follows.

- **Section 1** describes the context of the document and provides an introduction to the project.
- **Section 2** discusses the adopted software engineering process and presents the SCRUM organization.
- **Section 3** provides an overview of the functional and non-functional requirements alongside the design of the project and needed diagrams.
- **Section 4** describes the architectural and design choices.
- **Section 5** describes the implementation, the main functionalities, and how the technologies were implemented, as well as the distribution of assignments between the team members.
- **Section 6** discusses the verification and validation of functional and non-functional requirements, and how the tests have impacted the deployment process.
- **Section 7** provides links to source code and deployed product.
- **Section 8** provides an in-depth analysis of the time and resources invested in the project.
- **Section 9** endorses the main lessons learned from the process of working on the project.

2 Adopted Software Engineering Process

In order to successfully build the PrivTAP system, the team decided to adopt an Agile methodology, more specifically SCRUM. This choice proved to be successful throughout the entirety of the project, enabling flexible task tracking and simplifying coordination between team members who operated in a geographically distributed context.

2.1 Team Organization and Roles

Right at the beginning of the project, two sub-teams were created to independently handle the frontend and backend. The frontend team was composed of 3 developers, Basic Enna, Benedetti Michele, Radonjic Ivan meanwhile Panseri Leonardo, Santoro Emanuele, Sassi Alessandro and Vaccarini Lorenzo Maria were in charge of the backend. This division was mainly based on previous knowledge and personal choices, in order to increase efficiency while developing and to minimize overheads. In addition, even though this could have potentially represented communication problems within the entire team, in the end, this was not the case, since the team members continued to have weekly meetings which included the participation of every member of the team. Furthermore, to deal with the partitioning of duties, detailed API documentation using Swagger was developed,

so as not to face the risk of inconsistent calls between the two modules.
The team also agreed on having a rotating SCRUM Master, meaning that in each Sprint a different member of the team would be assigned a SCRUM Master role. In retrospect, the team agrees that it was a great practice since it made everyone take an active role in management activities that would have otherwise been limited to a single team member. Finally, the Product Owner (PO) role is held by Ivan Radonjic.

2.2 SCRUM Process Overview

The project was divided into a total of 7 Sprints. The initial one, called "Sprint 0", lasted 1 week and was used by the team to agree on the initial project plan and architecture, as well as for defining the roles of each team member. All subsequent Sprints, each lasting 2 weeks, focused on product-related tasks, such as requirements definition, system design and implementation. Such a short Sprint duration gave us more control over the development process since the team members were able to plan ahead which tasks had to be completed during each Sprint in a more accurate way.
Due to the high variability in each team member's schedule, daily stand-up meetings were held either via text chat or group call on Discord, and each team member was exposed to whether they were encountering any problems while carrying out their work.

Meetings and Strategies Regarding meetings, the team always held a periodic, bi-weekly General Meeting the day before the start of each Sprint, encompassing both the Sprint Review and Retrospective and Sprint Planning activities. This moment has proven to be particularly useful for all team members since it provided them with a clear overview not only of what to gradually improve at each and every step of the process but also of the roadmap for the immediate future.
Weekly Supervisor Meetings were held in order to make sure that all activities in the development process were following the correct schedule without being excessively delayed. These meetings were usually followed by a Group Meeting to make sure every member was up to date with respect to how the overall work was proceeding, as well as to incorporate any suggestions pointed out by the supervisors.
Finally, Pair Programming was often adopted between group members to speed up the development while encouraging knowledge sharing, so that all team members working on the same or connected components were always on the same page.

2.3 Task Management

Task management is an essential part of the SCRUM process, and thanks to Jira team members managed to handle it fairly smoothly [2]. Jira also allowed the team to more easily follow the SCRUM conventions and processes since all work had to be split up per Sprint and all tasks had to be included in the product backlog before any team member could start working on them.
In addition, the team adopted a convention where each VCS commit had to explicitly link in its message the Jira task(s) that it referred to, in order to easily map its impact on the project. A full description of the convention can be found here: Conventional Commits.
The following process was followed throughout the project for task management:

- During the bi-weekly General Meeting the team brainstormed the tasks to be added to the Product Backlog, together with an estimate of the time required to complete them;

- At the end of the meeting, the Product Owner added all tasks in the Jira backlog, entering a Title, an optional Description, the estimated duration in **Story Points**, its priority, and the list of assignees who would be in charge of the task;
- During each Sprint, tasks were organized using 4 columns: *ToDo*, *In Progress*, *In Review* (for tasks that needed additional review by other team members before being committed), and *Done*;
- A task was moved to the *In Progress* column whenever one of its assignees started working on it and was marked as *Done* whenever the assignees finished the task and all its child tasks (if present);
- Story Points for each task were updated by task assignees to provide an always up-to-date view of the project advancement.

Furthermore, the team also kept track of each individual's working hours by using an internal spreadsheet, where each member would annotate an estimate of the hours they spent working on various categories of tasks, such as Documentation, Code and Meetings.

2.4 Verification and Validation Activities

In order to ensure that the product respected both the functional and non-functional requirements identified during the process, the team used a number of Software Verification and Validation techniques, both automated and manual:

- **Automated Tests** were written by the developers of each module and automatically executed whenever a Pull Request was made from a development branch onto either *main* or *production* branches. These tests were aimed at checking the behaviour of each module separately (Unit Tests), but also how it integrates with other components of the system (Integration Testing), and were created for both the back-end (server-side components) and front-end;
- **Code Review** was performed whenever a team member created a Pull Request from a development branch onto the *main* or *production* branches, to avoid either wrong or untested code would be released in these branches;
- **Acceptance Tests**, written by the team and applied to the product to make sure that it would meet the defined acceptance criteria and verify the correctness of the functional behaviour of the system.

3 Requirements

The following sections will describe and provide a brief explanation of both functional and non-functional requirements identified for the PrivTAP platform.

3.1 Functional Requirements

Table 1 shows the high-level functional requirements that the team elicited for the applications. The process of writing and choosing such requirements was based both on the original project proposal and on the communication with the sponsor. Most of these requirements focus on the actions that users can perform in order to use the platform, as well as on privacy configuration options, a core part of the project. Finally, the column "Satisfied" reports which requirements were fully

implemented in the final product.

Identifier	Requirement Name	Description	Satisfied
FRE1	Registration	A new user should be able to register on the platform.	✓
FRE2	Login	An existing user should be able to log in using his/her email address and password.	✓
FRE3	Create Service	An OSP should be able to create services.	✓
FRE4	Read Service	An OSP should be able to Read created services.	✓
FRE5	Update Service	An OSP should be able to Update created services.	✓
FRE6	Delete Service	An OSP should be able to Delete created services.	✓
FRE7	Create Trigger	An OSP should be able to publish one or more triggers that are connected to a possible event happening on their side.	✓
FRE8	Read Trigger	An OSP should be able to read all the created triggers associated with a Service where the OSP is the owner.	✓
FRE9	Update Trigger	An OSP should be able to update triggers associated with a Service where the OSP is the owner.	✓
FRE10	Delete Trigger	An OSP should be able to update triggers associated with a Service where the OSP is the owner	✓.
FRE11	Manage Action	An OSP should be able to provide one or more actions that permit the platform to make operations on its side.	✓
FRE12	Read Action	An OSP should be able to read all the Actions associated with a Service of which it is the owner of it.	✓
FRE13	Update Action	An OSP should be able to update Actions associated with a Service of which it is the owner of it.	✓
FRE14	Delete Action	An OSP should be able to delete Actions associated with a Service of which it is the owner of it.	✓

FRE15	Create Privacy Level Configuration Options (Permissions)	A service offers a configuration option for required data needed to perform a trigger or an action, in order to provide fine-grained privacy options configurable from the platform. This option is called permission. An OSP should be able to create a Permission that can be associated with a Service.	✓
FRE16	Read P.L.C.O. (Permissions)	An OSP should be able to read all the Permissions created.	✓
FRE17	Update P.L.C.O. (Permissions)	An OSP should be able to update Permissions created.	✓
FRE18	Delete P.L.C.O. (Permissions	An OSP should be able to delete Permissions created.	✓
FRE19	Rule Creation	A logged user should be able to create trigger-action rules specified as "if <trigger> then <action>" using the authorized trigger or action and compatible with each other. An OSP should be able to delete Permissions created.	✓
FRE20	Rule Deletion	A user should be able to delete a rule they created.	✓
FRE21	Explore Services	A user should be able to see a list of available services that they want to use for their automation rules and check the authorized ones.	✓
FRE22	Explore Trigger	A user should be able to see the list of all the triggers exposed by a service and the permissions required in order to use the trigger in a Rule.	✓
FRE23	Explore Action	A user should be able to see the list of Actions exposed by a service and the permissions required in order to use the action in a Rule.	✓
FRE24	Explore Rule	The user should be able to see all the already created rules.	✓
FRE25	Service Authorization and Granularity	A user should be able to check the granularity level offered by the service and authorize only selected permissions associated while integrating the service within the platform.	✓
FRE26	Error Reporting	The system should report to the user whenever an error occurs.	✓
FRE27	Rules execution	All the actions in the rules defined by the user should be activated when the corresponding trigger event occurs.	✓

Table 1: Functional Requirements

3.2 Non-functional Requirements

Table 2 shows the non-functional requirements that the team produced for the application. Although these requirements were not explicitly mentioned in the project proposal, the team thought it was important to take them into account in order to grant more usability to the system.

Identifier	Requirement Name	Description	Satisfied
NFRE1	Login Response Time	The system should be able to respond to a user login action within 2s from when the user presses the Login	✓
NFRE2	SignUp Response Time	The system should be able to respond to a user sign-up action within 3s from when the user presses the Sign Up button	✓
NFRE3	General Response Time	To any general action, the system should provide feedback to the user within 5s	✓
NFRE4	System Security	The system should provide password hashing and API authorization and authentication so that only registered user can access their personal pages.	✓

Table 2: Non Functional Requirements

In addition, given that the focus of the project was to build a privacy-oriented platform that allows the user to check which type of data is used when their rules are being executed, one major constraint was about authorising PrivTAP to access the user's personal data on external Services through OAuth 2. In particular, the token request was designed with the new OAuth2 Draft [14], which provided a flexible way to implement permission granularity levels.

4 Design

The sections below will outline and briefly explain the architecture of the platform and the key design choices made.

4.1 Overview

The application is a three-tier architecture as shown in Figure 1. From a high-level perspective, the overview includes three main blocks:

- Vue application (frontend)
- NodeJS application (backend)
- MongoDB (database system)

In particular, we chose the Representational State Transfer (REST) architectural style to have high decoupling of frontend and backend with (mostly) stateless, cacheable interactions between the two of them, and to follow the de-facto standard for web-based systems.

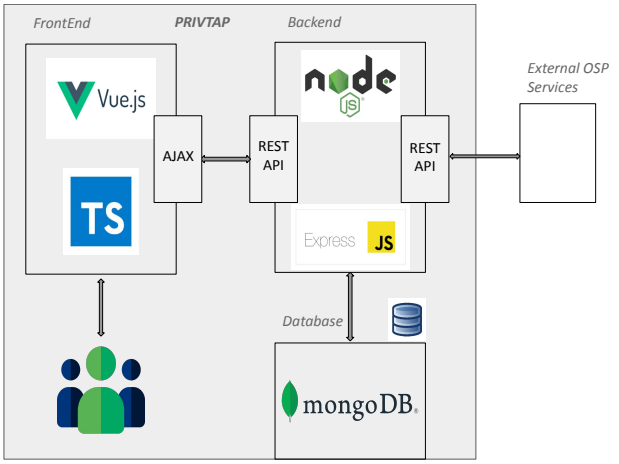


Figure 1: High-Level Overview of the Platform

4.2 Architecture Details

In figure 2 there is the overview of the components needed for the backend application. Each component has a set of endpoints that can be used by external users to communicate with it, as long as the requests follow the specific endpoint interface documentation. This documentation can be found on our platform and it will explain how to communicate with each endpoint. [15]

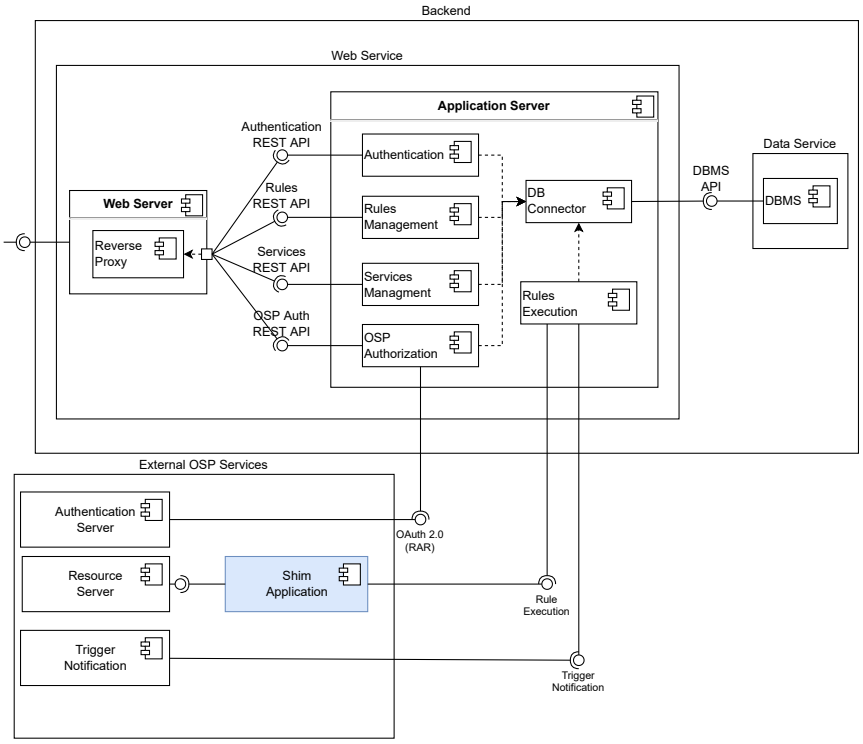


Figure 2: The component diagram of the backend application

As explained in the component diagram, the PrivTAP backend handles a variety of API routes, which are used to implement the platform behaviour as well as access control and authentication. In addition, some of these routes are built to handle communication with external OSPs for tasks such as OAuth 2.0 authorization and data extraction/delivery for rule execution.
An external OSP can integrate itself with the PrivTAP platform by building a Middleware (Shim Application Component) in order to properly communicate with PrivTAP. In particular, the team defined a specific data exchange format for Rule Execution and the purpose of the Middleware is to translate from the generic OSP interface to our standard format. Besides that, the Middleware must also be able to filter the data sent to PrivTAP by limiting it to the bare minimum required for the action execution. In this way, only data that are needed by the action reach our platform and we avoid data leaks. More about the Middleware can be found in the OpenAPI Documentation, in the "Middleware" section. [15]

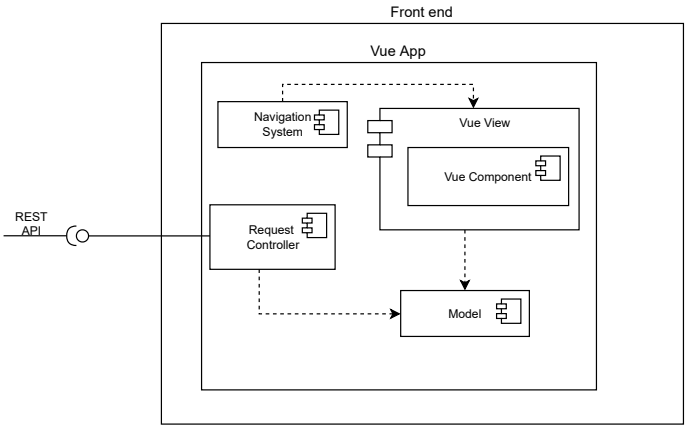


Figure 3: Component diagram of the frontend application

Figure 3 illustrates the main components of the frontend, which is developed as a single-page application. The MVVM pattern [26] was employed to ensure a fast response of the view after data updates.
Finally, Figure 4 shows an overview of the database architecture of PrivTAP. As evidenced by it, services are created by OSPs and group triggers and actions that share common properties, such as the endpoint base URL and the OAuth path. Each trigger or action has a set of permissions associated with it, which are defined by the OSP and need to be authorized by end-users in order to integrate and use them for the creation of a rule.

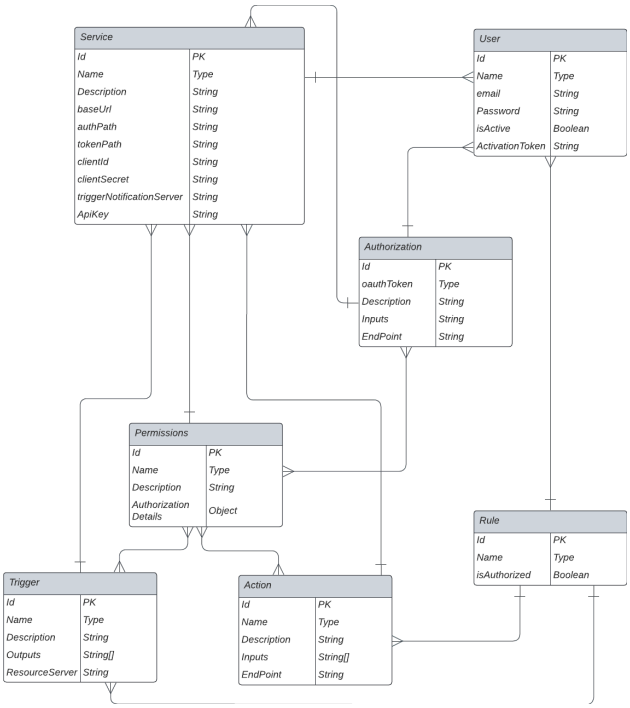


Figure 4: The ER schema of the database

5 Implementation

This section focuses on illustrating the ways and aspects of how the team faced and dealt with the challenges posed by the implementation of the project.

5.1 Technologies Used

The members of the team had different knowledge and development backgrounds, hence each member had to learn and familiarize themselves with both the technologies and programming languages used, which are presented below. Nevertheless, such instruments provided valuable support both for organizing the teamwork and schedule and for making the development of complex features and sub-systems easier.

5.1.1 Tools

- **Jira** [2], mainly used for handling the product/Sprint backlog and for the allocation of tasks between team members. It has been beneficial in every aspect for all team members, especially for its useful option that allows the generation of burndown charts automatically from the state of tasks in the Sprint backlog.
- **Git and GitHub**[3], utilized for version control of the project.
- **GitHub Actions**[22], adopted for CI/CD integration and configured to automatically deploy the system every time a new commit was pushed on the production branch if and only if all the tests were passing.
- **Swagger**[4], used to document the platform API.
- **Discord**[9], the key communication tool for team members regarding stand-up meetings, updates, general communication, arrangements and similar.
- **Overleaf**[10], utilized to handle project reports.
- **Excalidraw**[11], useful for defining draft sketches of diagrams such as UML diagrams, component diagrams and similar.
- **Google Meet**[12], mainly used for meetings with our supervisors.
- **Google Docs**[13], utilized for note-taking regarding meetings (supervisions, team, stand-up), as well as for writing all course-related presentations.
- **Google Drive**, used to archive the project documentation while it was being written.

5.1.2 Hosting Platforms

To be able to successfully deploy our deliverables as a Cloud platform and host our full code base, the team chose the following services:

- **GitHub**[5], to host the Git repository [25].
- **Google Cloud** [23], to host the deployed PrivTAP platform.
- **MongoDB**[24]for the platform Cloud-hosted database.

5.1.3 Programming Languages and Libraries

In order to implement the complete application the team decided to use the following frameworks and programming languages:

- **TypeScript**[18], used for both the frontend and backend applications as it grants improved code completion, stronger type checking and better integration with IDEs.
- **NodeJS**[8], needed for the backend, in order to have a framework for the REST architecture.
- **Vue.JS**[6] as a framework for building the frontend application.

5.2 The Product

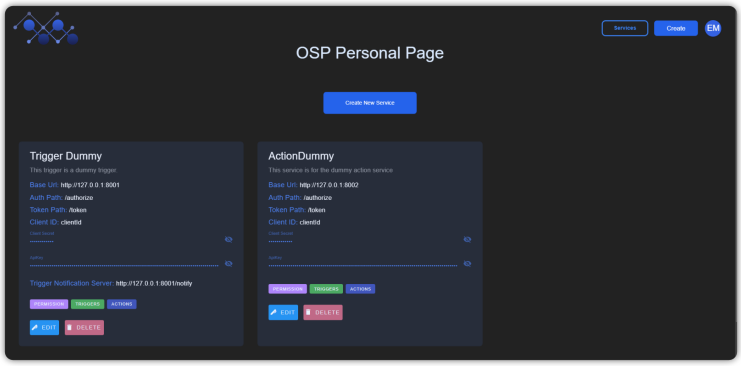


Figure 5: Screenshot of the page which lets the OSP add, edit or remove service

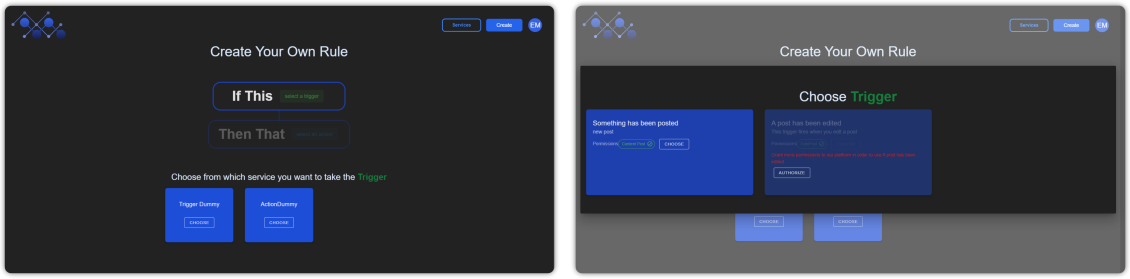


Figure 6: Screenshots of the page that lets the user create a new Rule

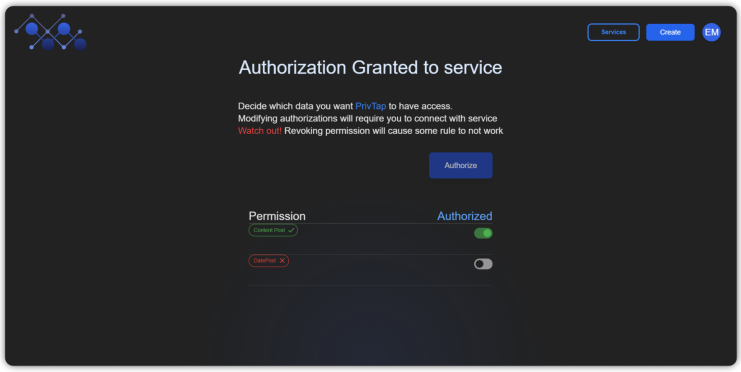


Figure 7: A screenshot of the page that lets the user grant fine-grained permissions for a specific service

5.3 Deliverables

The deliverables produced by the team are the followings:

- **PrivTap Platform**: The PrivTap platform is composed of a Single Page Application frontend that interacts with its associated backend through REST API calls.
- **Dummy Trigger OSP**: The implemented trigger OSP is the mockup of a simple external service that integrates with the PrivTap platform. Its purpose is to let users create and publish posts, and the associated trigger is fired when the users create a post, and with OAuth 2.
- **Dummy Action OSP**: The implemented action OSP is the mockup of a simple external service that integrates with the PrivTap platform. Its purpose is to let users create and publish posts, and the associated action is the creation of a post for a specific user.

5.4 Installation

To be able to run the project locally, you will need to download its source code from the GitHub repository [25]. You will then need to follow the instructions listed below to start the various modules. A deployed version of the platform is already available at `https://privtap.it`.

5.4.1 Backend Installation

The backend application requires Node.JS to be installed on your machine. After you have installed Node.JS, follow these steps to run the application:

1. Open the "backend" directory in the PrivTAP project folder, create a file named ".env" and fill it with the text below (make sure to replace the data in quotes with your own):
```
PORT=8000
BASE_URL=/api/
NODE_ENV=development
SALT_ROUNDS=8
JWT_SECRET="insert_secret_string"
JWT_EXPIRE="insertExpDateInMs"
DEPLOYMENT_URL=https://privtap.it
```

```
FRONTEND_URL=http://localhost:5173
DB_STRING="your_MongoDB_URL"
MAILJET_SECRET_KEY="your_mailjet_key"
MAILJET_SENDER="your_email_sender"
```

2. Start a terminal window on your machine, and type `cd <project_directory>/backend`.
3. Run the following commands:
   ```
   npm install
   npm run run
   ```

5.4.2 Frontend Installation

Just like the backend, the frontend application requires Node.JS to be installed on your machine. You can then run the application with these steps:

1. Start a terminal window on your machine and type `cd <project_directory>/frontend`.
2. Run the following commands:
   ```
   npm install
   npm run dev
   ```

6 Verification and Validation

To ensure that the Project complied with the previously mentioned requirements and that the product was adequate for its intended use, continuous verification and validation were crucial steps in our software development process.
For each component and module that is present on the Backend and Frontend systems, *Unit Tests* and *Integration Tests* were implemented starting at the very beginning to keep track of the state of the implemented functionalities, related to the specified requirements.

6.1 Functional Requirements

In order to verify and validate our functional requirements the team developed automated test case suites that ran every time a branch was merged into main or production branches. In particular, we used external libraries that allow mocking and stubbing of functions or modules that are not related to the component that needs to be tested, such as MongoDB calls, HTTP call, and external functions call. The two libraries used for this aim were *Sinon* [19] and *Mocha* [20]. A brief summary of all unit/integration tests executed is presented in the section below, along with the coverage achieved for each endpoint in the case of backend tests and module or component in the case of frontend tests.

6.2 Unit Tests

On both the backend and frontend of the system the team developed an extensive set of automated unit tests that aim to cover both normal execution paths and corner cases, to minimize the risk of total system failures when deployed. Such unit tests proved to be useful under many circumstances where the theoretical or designed module behaviour did not match its effective output, thereby helping to identify and fix bugs in the code.

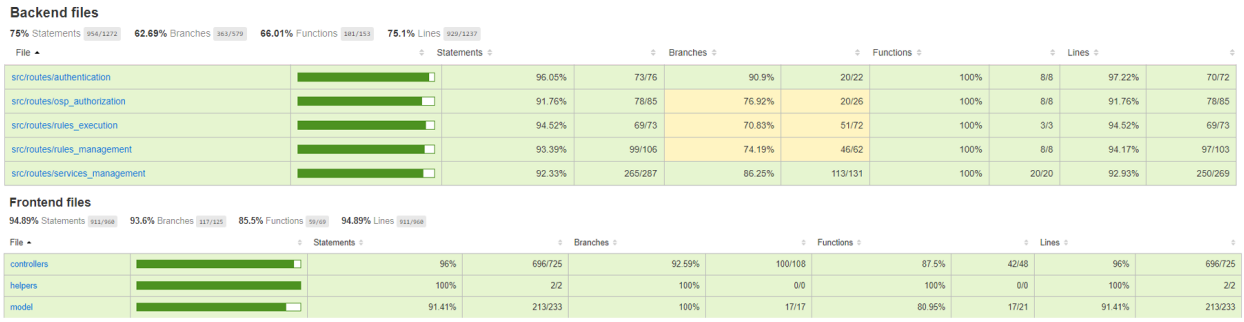
6.3 Integration Tests

Together with unit tests, Integration tests were developed to check that all written sub-modules, such as internal ones or helper API routes, behaved as expected when cooperating together. Such tests allowed the team to verify the behaviour of the system on a higher-level scale, closer to how the backend could have been used by the frontend module.
The following example shows how, after stubbing the MongoDB insertion method, the test performs a call to the login endpoint with some dummy user data and it checks that the response reported a successful operation.

```
1 it ("should succeed when all the parameters are well defined", async () => {
2     insertNewUserStub.resolves(true);
3     const res = await requester.post("/register").send({ username : "someUsername"
          , email: "someEmail@gmail.com", password: "somePassword" });
4     expect(res).to.have.status(200);
5 });
```

Listing 1: An example of Integration Test for the Login module

https://www.overleaf.com/project/638675eee2c07a66f89582f8



Backend files

75% Statements 954/1272 **62.69%** Branches 363/579 **66.01%** Functions 101/153 **75.1%** Lines 929/1237

File	Statements		Branches		Functions		Lines	
src/routes/authentication	96.05%	73/76	90.9%	20/22	100%	8/8	97.22%	70/72
src/routes/osp_authorization	91.76%	78/85	76.92%	20/26	100%	8/8	91.76%	78/85
src/routes/rules_execution	94.52%	69/73	70.83%	51/72	100%	3/3	94.52%	69/73
src/routes/rules_management	93.39%	99/106	74.19%	46/62	100%	8/8	94.17%	97/103
src/routes/services_management	92.33%	265/287	86.25%	113/131	100%	20/20	92.93%	250/269

Frontend files

94.89% Statements 911/960 **93.6%** Branches 117/125 **85.5%** Functions 59/69 **94.89%** Lines 911/960

File	Statements		Branches		Functions		Lines	
controllers	96%	696/725	92.59%	100/108	87.5%	42/48	96%	696/725
helpers	100%	2/2	100%	0/0	100%	0/0	100%	2/2
model	91.41%	213/233	100%	17/17	80.95%	17/21	91.41%	213/233

Figure 8: Overview of the test coverage report

Overall the project has a total of 250 tests, all passing.

6.4 Non Functional Requirements

Non-Functional requirements are often as important as Functional ones because they can have a significant impact on the overall quality and effectiveness of the system, and for this project, the team decided to focus on the ones related to Security and Performance.
For the Security part, the main goal was to keep the user data protected from possible data breaches by hashing all sensitive information and preserving access to our endpoints only to authorized users that effectively have the correct permission to access it. In order to implement this functionality an external package called *bcrypt* [21] was used as shown below:

```
1 const hash = hashSync(password, env.SALT_ROUNDS);
```

Listing 2: Password Encryption

On other hand, the Performance validation of the specified requirements was made by implementing tests using the Postman Test Tools [17] in order to make sure that API endpoints had an acceptable

response time. The script below shows the test written and used by Postman to evaluate the response time of a tested endpoint.

```
pm.test("Response time is less than 2000ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(2000);
});
```

Listing 3: Postman Test

The figure below shows the test result for the login endpoint with its relative response time.

Figure 9: Login response time endpoint tested with Postman

Finally, the Lighthouse tool was also used to collect and analyze performance metrics of the web application, which helped the team identify bottlenecks when creating or refreshing pages after user actions. The metrics provided by the Lighthouse extension are displayed in the following image.

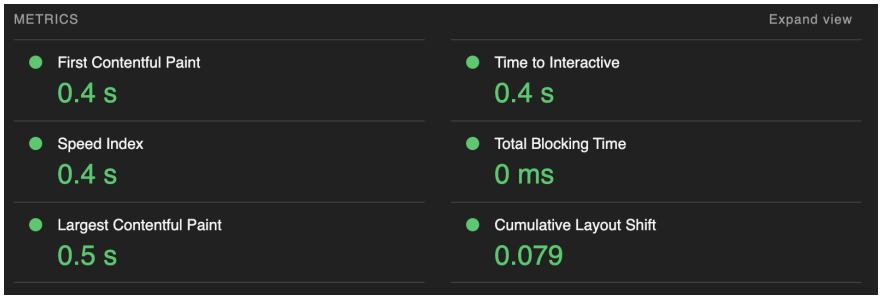


Figure 10: Perfromance metrics given by lighthouse analysis

7 Demo

The following section will provide the needed URL for the website so anyone can access it freely, as well as the URL for the GitHub, so that anyone can access and see the product themselves.

7.1 Website

Link to the final product: `https://privtap.it`.
Demo link: `https://youtu.be/0a2KSywWvF4`

7.2 GitHub

PrivTAP Github link: `https://github.com/PrivTap/PrivTap`

8 Project Effort

This section provides an in-depth analysis of the time and resources invested in the project, including breakdowns of tasks and responsibilities.

8.1 Effort Overview

The project lasted 3 months, which resulted in a total of around 1200 man-hours and an average of 92 hours/week spent by team members. This figure breaks down into a total of 344 hours spent during meetings, 106 hours for documentation, 630 hours for development-related activities, and 44 hours for other uncategorized tasks.

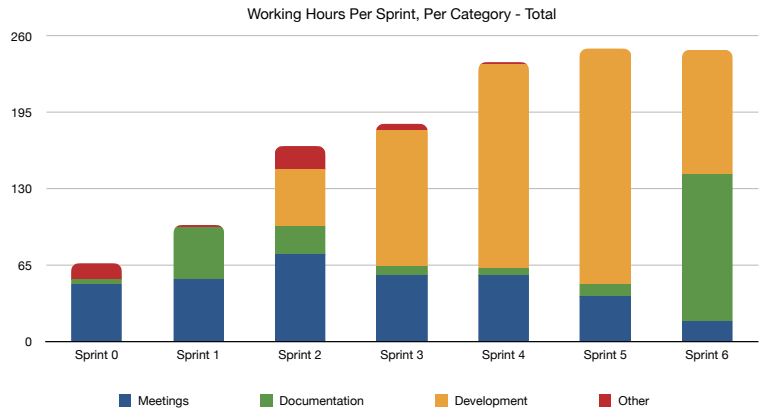


Figure 11: Breakdown of the total work hours by Sprint and Category

The chart shown in figure 11 illustrates how the focus of each Sprint changed over time: initial Sprints were mostly made up of long meetings since the team had to agree on core aspects of the project such as the overall architectural design and technology stack, as well as documentation to build a solid foundation for development tasks. Starting from Sprint 2, however, the focus gradually shifted towards development, with later Sprints being mostly made up of development activities and shorter meetings.

8.2 Sprint Burndown Charts

As you can notice from the Figure 12 below, in the first sprint the team encountered some problems and a lack of knowledge related to how Jira works internally, which therefore required a learning period after which task duration estimation started working as expected. In addition, during the Sprint 3 Retrospective, an underestimation issue of tasks emerged, and it was fixed in the next Sprints. Finally, due to bank holidays, the team members had a long, flat period in the middle of the last Sprint, but still managed to finish vital tasks in time.

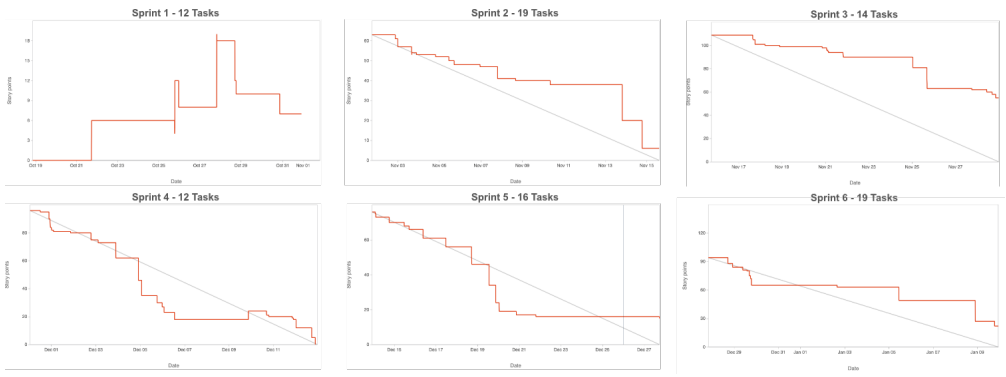


Figure 12: Jira Burndown Charts

8.3 Workload Organization

The graph in Figure 13 shows how the workload (in terms of effective working hours) was divided among team members meanwhile, Figure 15 as well as a breakdown of the hours spent by each member on a per-Sprint basis.

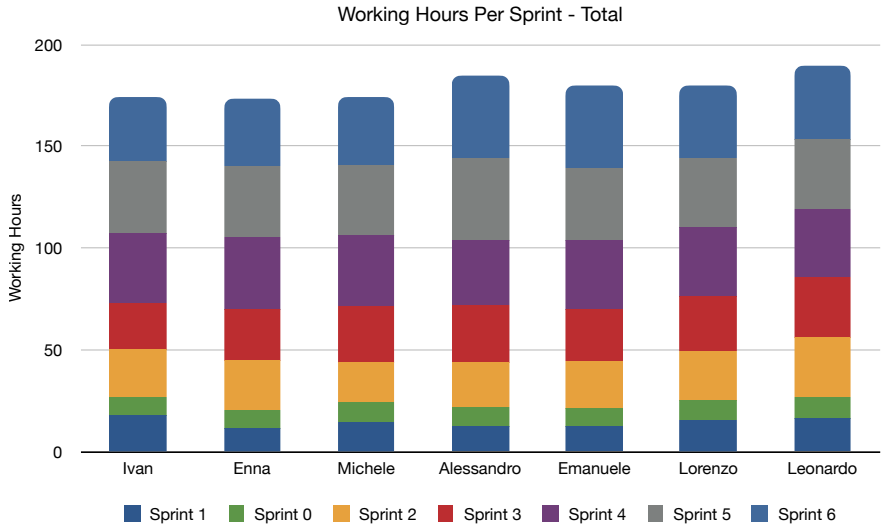


Figure 13: The total number of hours spent by each team member, per Sprint

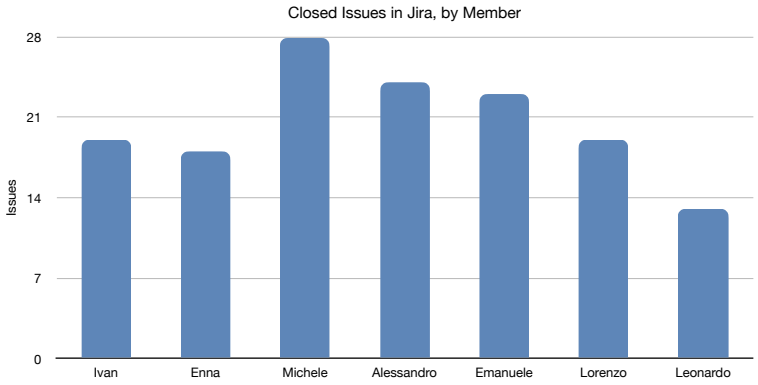


Figure 14: The total number of closed issues by team member

Overall, the team managed to achieve a fairly uniform distribution of the workload, with differences being related mainly to the task each team member was assigned to for each Sprint. A possible improvement would have been to divide and assign tasks in a more efficient way by estimating more accurately their time to avoid having such discrepancies.

8.3.1 Other Project Metrics

The following chart illustrates the total number of commits (excluding Pull Requests) for each team member. Even though high discrepancies are evident, this is mostly due to the pair programming practice, which at times made one member commit to work that was done by the other teammate.

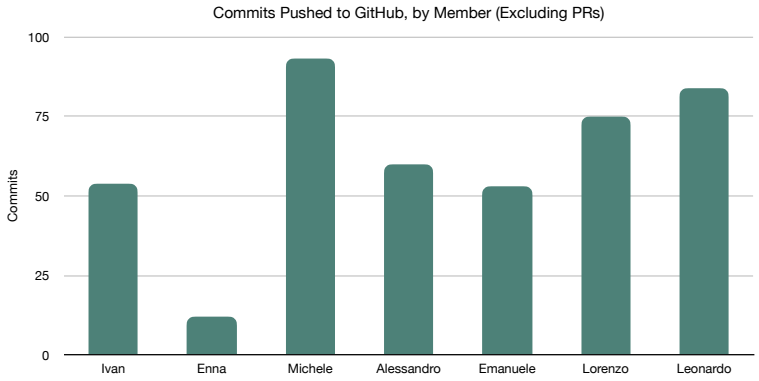


Figure 15: The total number of commits pushed to GitHub, by Member

9 Retrospective and Lessons Learnt

9.1 Positive Experiences

The project allowed team members to learn about different aspects of the distributed development process. Overall, team members reported a positive experience which brought them to learn a multitude of skills applicable even outside the scope of such a project. Some key highlights of what the team considered positive are:

- **Communication**: All members learned how to communicate in an effective and professional way, taking into account both cultural differences and geographical distance.
- **Requirements Engineering**: facing such an articulated project required the systematic approach typical of Requirements Engineering and Software Engineering.
- **System Design Phase**: having an initial design phase helped the team understand which technologies and architectures had to be studied and implemented.
- **New Technologies**: The architectural design phase gave the team members the opportunity to discover and try new technologies which, despite being widely used in industrial and commercial applications had not been previously used by team members. Some examples include NoSQL Databases, REST API design and implementation, and the Vue framework.
- **Deployment to Cloud Services**: The automated deployment process ensured a better understanding of Cloud-based technologies and processes present on the market.
- **OAuth Protocol**: The nature of the project allowed the team members to experiment with the OAuth authentication protocol, implementing the new *Rich Authorization Request* IETF draft.

9.2 Possible Improvements

At the end of the project, the team also identified a series of points as possible improvements for future projects. In particular, the team identified:

- **Workload Estimation**: Since most of the defined tasks were conceptually new to team members, some difficulties arose regarding the actual workload estimation at the beginning of some Sprints, which resulted in having more tasks than what was feasible.

- **Low-Priority Functionalities**: Due to the complexity and time constraints of the project some of the originally planned, low-priority optional features could not be implemented. Although these features do not impact the main objective or overall functionality of the platform, they could have improved the user experience.
- **Development Planning**: To have a better overview of how the development process would have unfolded over the course of the project, a high-level schedule could have been developed during initial Sprints.

References

[1] https://www.scrum.org/

[2] https://www.atlassian.com/software/jira

[3] https://git-scm.com/

[4] https://swagger.io/

[5] https://github.com/

[6] https://vuejs.org/

[7] https://developer.mozilla.org/en-US/docs/Web/JavaScript

[8] https://nodejs.org/en/

[9] https://discord.com/

[10] https://www.overleaf.com/

[11] https://excalidraw.com/

[12] https://meet.google.com/

[13] https://docs.google.com/

[14] https://datatracker.ietf.org/doc/html/draft-ietf-oauth-rar-22

[15] https://privtap.it/api/docs/

[16] https://ifttt.com/docs

[17] https://learning.postman.com/docs/writing-scripts/test-scripts/

[18] https://www.typescriptlang.org/docs/

[19] https://sinonjs.org/releases/latest

[20] https://www.npmjs.com/package/mocha

[21] https://www.npmjs.com/package/bcrypt

[22] https://github.com/features/actions

[23] https://cloud.google.com/?hl=it

[24] https://www.mongodb.com/docs/

[25] https://github.com/PrivTap/PrivTap

[26] https://012.vuejs.org/guide/

[27] Mahadewa, K., Zhang, Y., Bai, G., Bu, L., Zuo, Z., Fernando, D., Liang, Z. and Dong, J.S., 2021, July. Identifying privacy weaknesses from multi-party trigger-action integration platforms. In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (pp. 2-15) https://baigd.github.io/files/ISSTA21-Taifu.pdf

[28] https://developer.mozilla.org/en-US/docs/Glossary/Shim?retiredLocale=it

[29] https://docs.google.com/document/d/1Vgr_Y_QO3EaVSOmiVQplwmikG7nE1kzoipYBCUqqo-0/edit