

# Project ADE Burn

Aymane Amessegher

David Darras

6th April 2025

## 1 Introduction

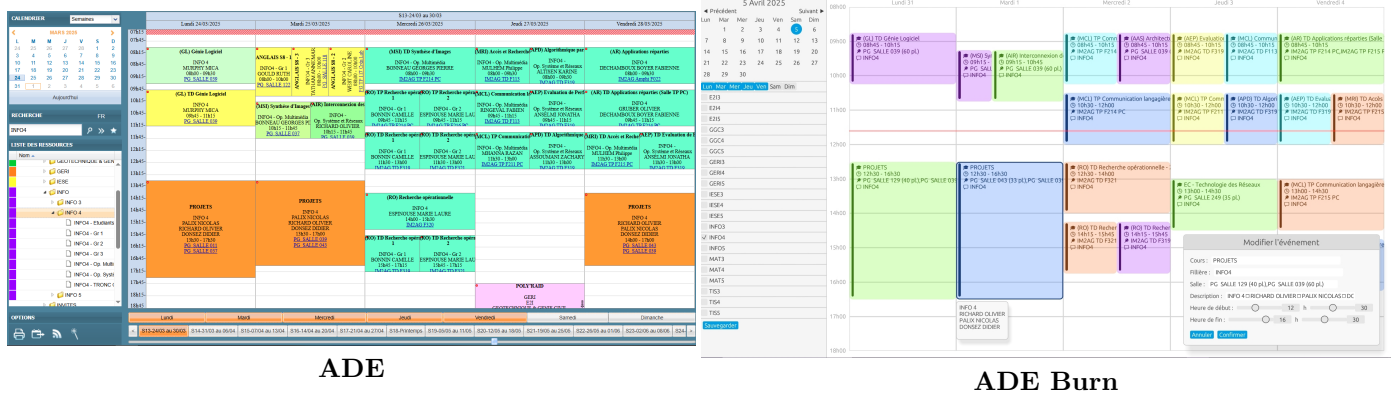
### 1.1 Context

The ADE application is a widely used tool in higher education institutions for managing timetables. It allows users to view schedules, assign rooms and resources, and adjust timings according to academic and administrative constraints.

However, one limitation of ADE is that it is not an open-source application, which means that its code is not accessible or customizable for institutions or developers who may wish to adapt or improve it. Because of this, we decided to create our own solution.

In our project, we were inspired by ADE to develop our own application in Rust, primarily using the **egui** library for the user interface. The goal is to offer a modern, high-performance, and interactive solution while benefiting from Rust's advantages in terms of safety and efficient resource management.

Our application incorporates some key concepts from ADE, such as dynamic timetable visualization and interactivity for modifying and adjusting schedules in real-time.



### 1.2 Basic concepts

- **Rust** is a language based on the concepts of ownership and borrowing, which offers many advantages: memory safety, no garbage collector, better concurrency management, etc. This is why this language was chosen for the project, in addition to the goal of learning a new programming language.
- **Cargo** is the package manager and build tool for Rust, used to compile, test, and manage project dependencies.
- **Egui** is a Rust library that allows you to create both heavy applications (i.e., desktop applications) with a graphical user interface and lightweight applications by compiling the code to WebAssembly, which can be run in a web browser.
- **Flask** is a Python web framework that allows you to easily create web applications. It provides basic features for creating a web server, such as URL routing, handling requests and responses, etc., based on a REST API (*Representational State Transfer*), allowing communication between our Python server and our desktop application via standard HTTP requests.
- **Request** and **Tokio** are crates that allow you to make HTTP requests and handle asynchronous operations in Rust in a simple and efficient manner.
- **Icalendar** is a Rust crate that allows you to build and parse calendars in iCalendar format. The files sent by the server are in the format: \*.ics and have the following structure:

```

1 BEGIN:VCALENDAR
2 METHOD:REQUEST
3 PRODID:-//ADE/version 6.0
4 VERSION:2.0
5 CALSCALE:GREGORIAN
6 BEGIN:VEVENT
7 DTSTAMP:20250331T101458Z
8 DTSTART:20250310T084500Z
9 DTEND:20250310T101500Z
10 SUMMARY:(GL) TD Génie Logiciel
11 LOCATION:PG SALLE 039 (60 pl.)
12 DESCRIPTION:\n\nINFO 4\nGénie logiciel\nMURPHY MICA\n(Exporté le:31/03/20
13 25 12:14)\n
14 UID:ADE6050726f6a65745547415f323032345f323032352d39363531392d382d30
15 CREATED:19700101T000000Z
16 LAST-MODIFIED:20250331T101458Z
17 SEQUENCE:2141707074
18 END:VEVENT
19 ...

```

These \*.ics files are stored in the directory `resources/backend/ics/` and they store events for a given week and a department, where the indicated date corresponds to the start of that week.

**Examples :** `INFO3_10_03_2025.ics`, `MAT4_24_03_2025.ics`, etc.

- An **event** refers to a class scheduled for a specific time period, in a specific room, with a designated instructor.

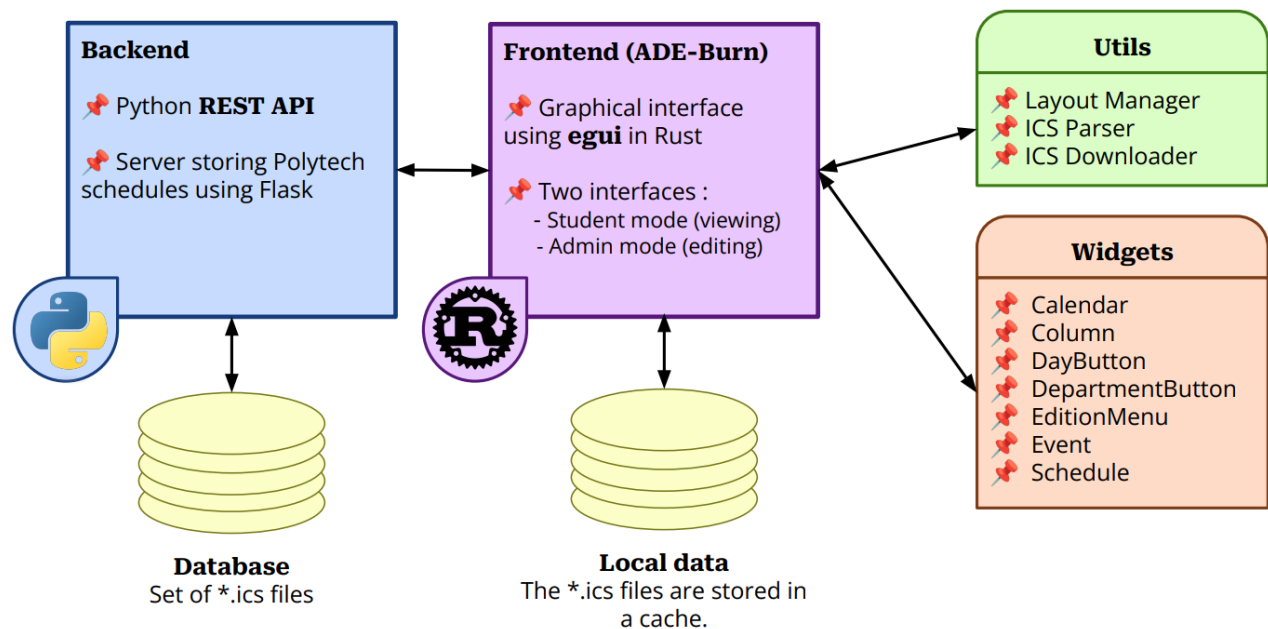
**Example :**

```

1 Date           : 14/03/2025
2 Cours          : (AR) Applications réparties
3 Filière        : INFO4
4 Professeur     : DECHAMBOUX BOYER FABIENNE
5 Salle          : IM2AG Amphi F022
6 Heure de début : 08h00
7 Heure de fin   : 09h30

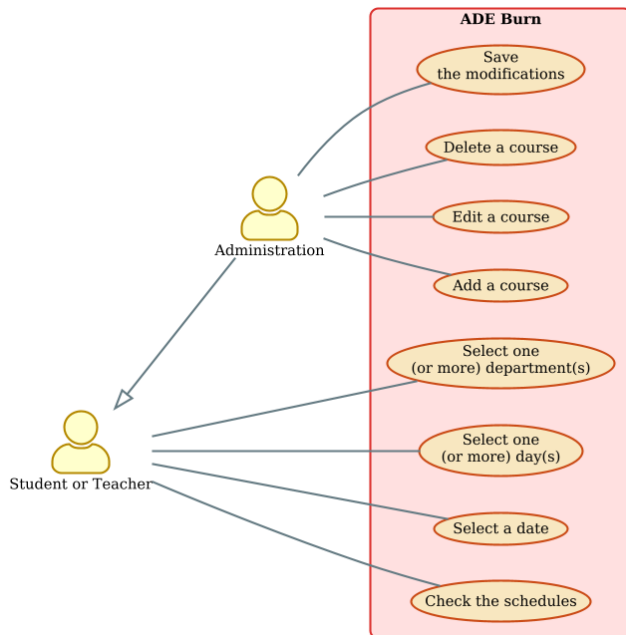
```

### 1.3 Application Architecture



## 2 Behavioral View

### 2.1 Use Case



- The main actors related to the *ADE Burn* system are the **students**, the **professors**, and the **administration** of Polytech Grenoble.
- **Save the modifications:** Press the button at the bottom left to locally save the changes made.
- **Delete a course:** Right-click on the event to be deleted.
- **Edit a course:** Left-click on the event to be modified.
- **Add a course:** Middle-Click on a column to add an event.
- **Select a department:** Check or uncheck the buttons on the left (INFO3, INFO4, etc.) to choose specific department events to display.
- **Select a day:** Press the buttons (Lun, Mar, Mer, etc.) to display only the days of interest. (Gray = disabled, Blue = enabled).
- **Select a date:** Click on a number in the calendar to change weeks. Use the Previous/Next buttons to navigate between months.

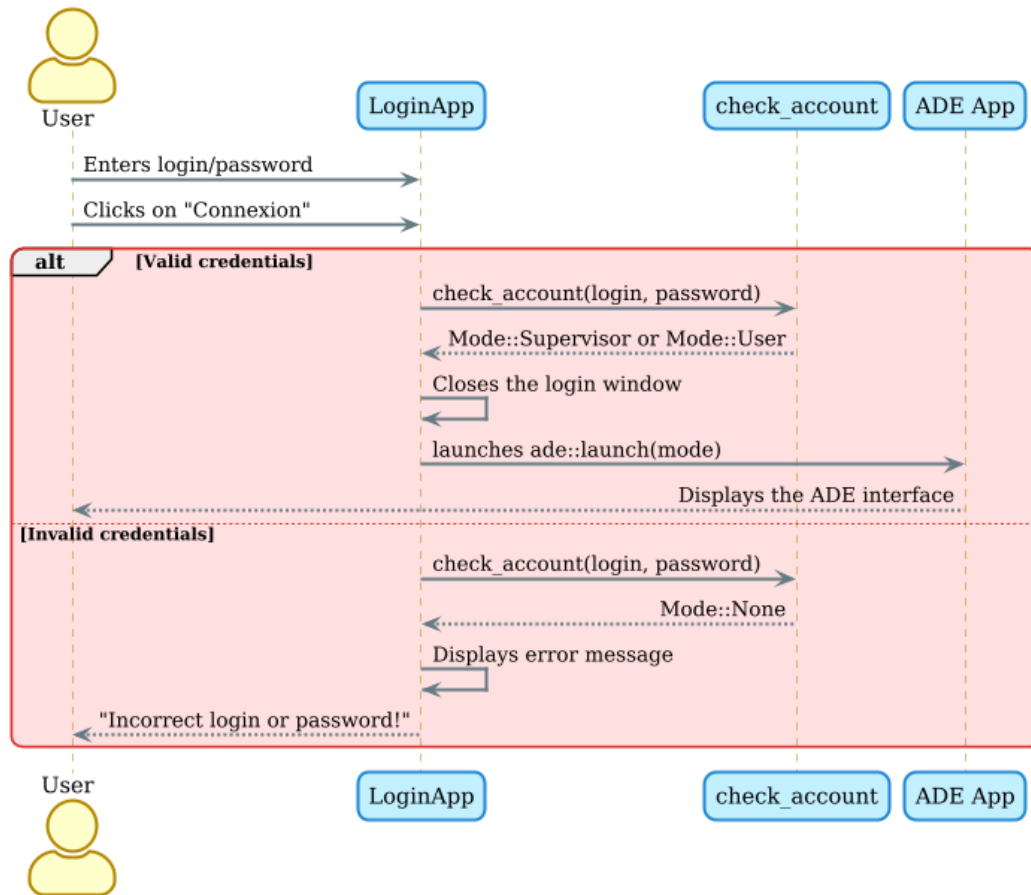
### 2.2 User Story: Typical Use Case for an Administrator

As a member of the administration, I want to manage and view the timetables of all students and professors, so that I can ensure the proper allocation of resources, avoid scheduling conflicts, and make adjustments as needed.

- **Login:** When I open the ADE Burn application, I am prompted to log in using my administrative credentials. After successful authentication, I gain access to the system where I can view and modify timetables for all students and professors.
- **View All Timetables:** Once logged in, I can view the timetables of all students and professors within the organization. I can easily switch between different departments, weeks, or individuals to ensure the schedules are accurate and up to date.
- **Modify Events:** If I need to make changes to the timetable (e.g., updating a class time, room, or instructor), I can click on an event to edit the details. I can adjust the event information and save the changes to update the timetable for the corresponding individual or department.
- **Add or Remove Events:** I can add new events by clicking on an empty slot or an unassigned time in the calendar. I can also remove events that are no longer necessary by right-clicking on them.
- **Save Changes:** After making adjustments to the timetable, I can save my changes, which will be applied to the system by clicking the "Sauvegarder" button at the bottom left.
- **Download ICS Files:** I can download ICS files for the timetables I'm working on. If the corresponding ICS file for a specific week is not available in the cache, I can request it from the server, and the system will notify me whether the download was successful or if there was an issue with the server.
- **Filter by Department or Discipline:** I can filter the timetable by department or discipline (e.g., INFO3, INFO4) to focus on specific student groups or classes. This helps me manage resources more effectively. However, I cannot display more than 100 events at once to maintain the application's fluidity and performance.

## 2.3 Connection to ADE

To be able to consult their timetable, the user, whether a student, a professor, or an administrative member, must log in with their *Agalan* credentials. In our code, we have only two hardcoded users within the application. We could have verified the individual's information through our server, but we chose to focus on the graphical interface rather than security issues. The two users are `user` and `root` with an eponymous password. To add more accounts, modify the `ACCOUNTS` table in `utils/account.rs`.



Connexion

GRENOBLE INP UGA POLYTECH' GRENOBLE

Identifiant universitaire :  
root

Mot de passe :  
....

Veuillez entrer votre identifiant et votre mot de passe.

Connexion

Connexion

GRENOBLE INP UGA POLYTECH' GRENOBLE

Identifiant universitaire :  
darrasd

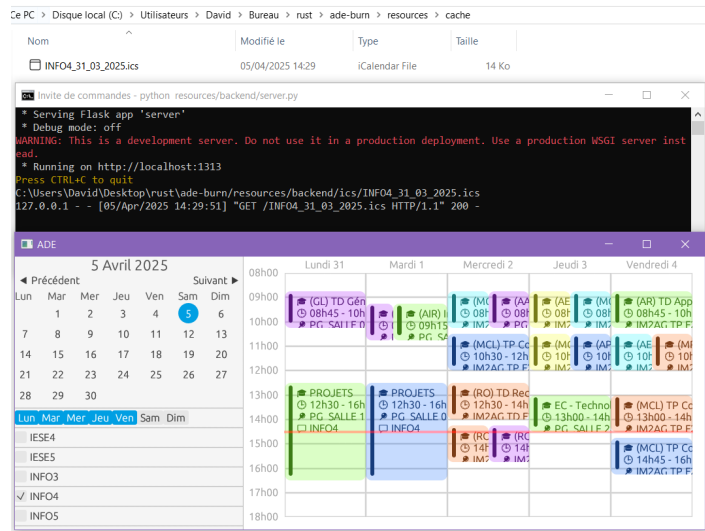
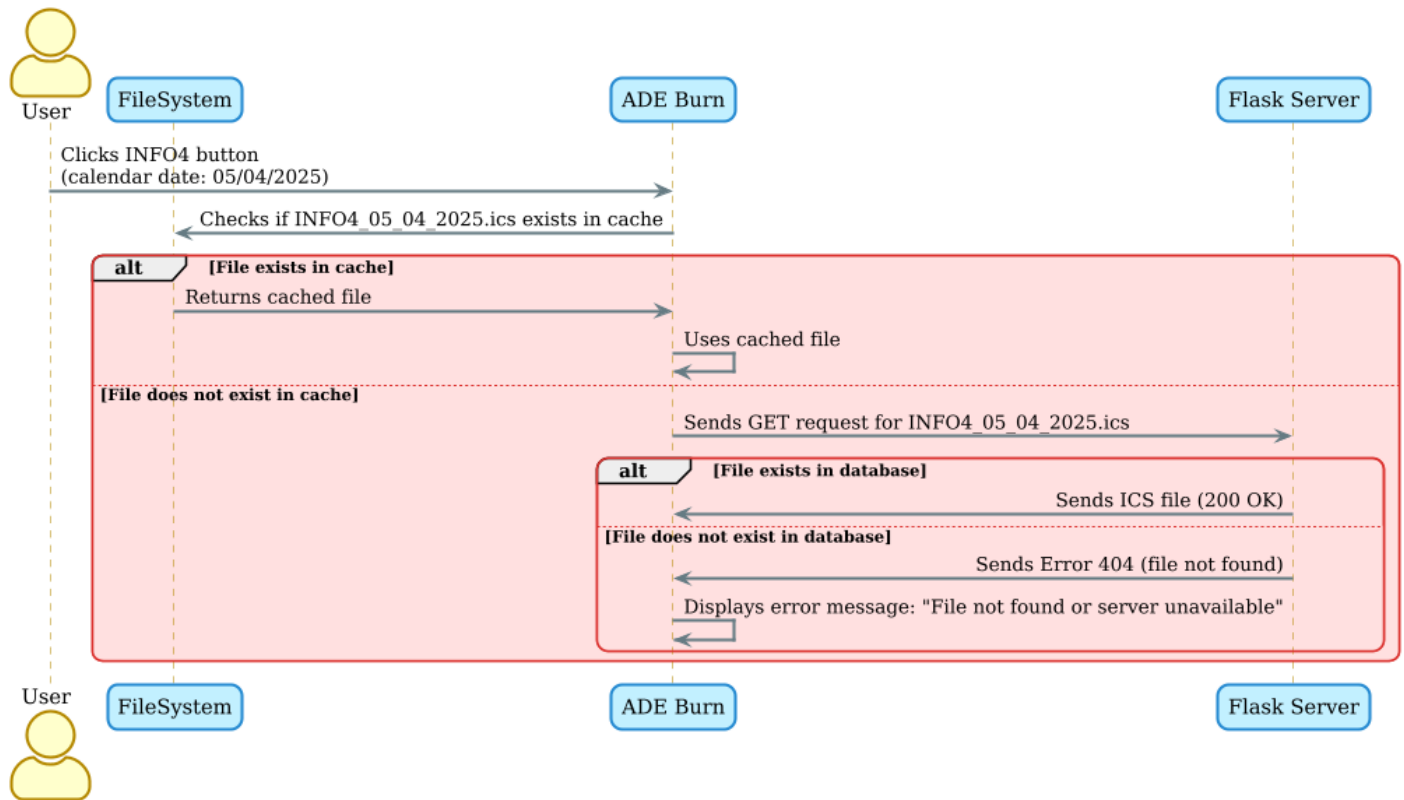
Mot de passe :  
.....

Identifiant ou mot de passe incorrect !

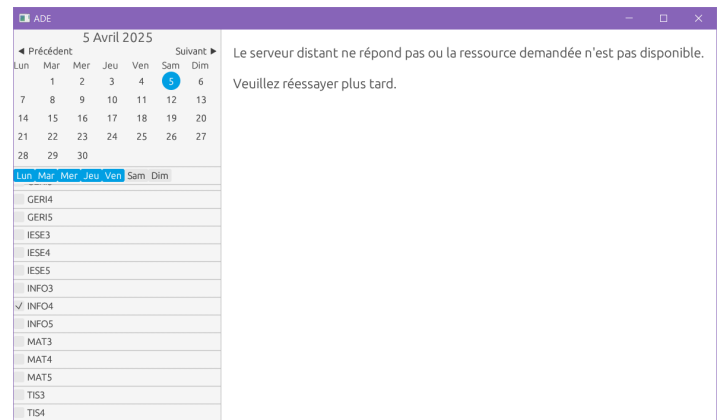
Connexion

## 2.4 Downloading an ICS file

When a user switches to a different week by clicking on the calendar or by adding/removing a discipline, the system checks if there is a corresponding file to update the events in the schedule cache. If not, it requests the file from the server. If the resource does not exist, the graphical interface notifies the user.



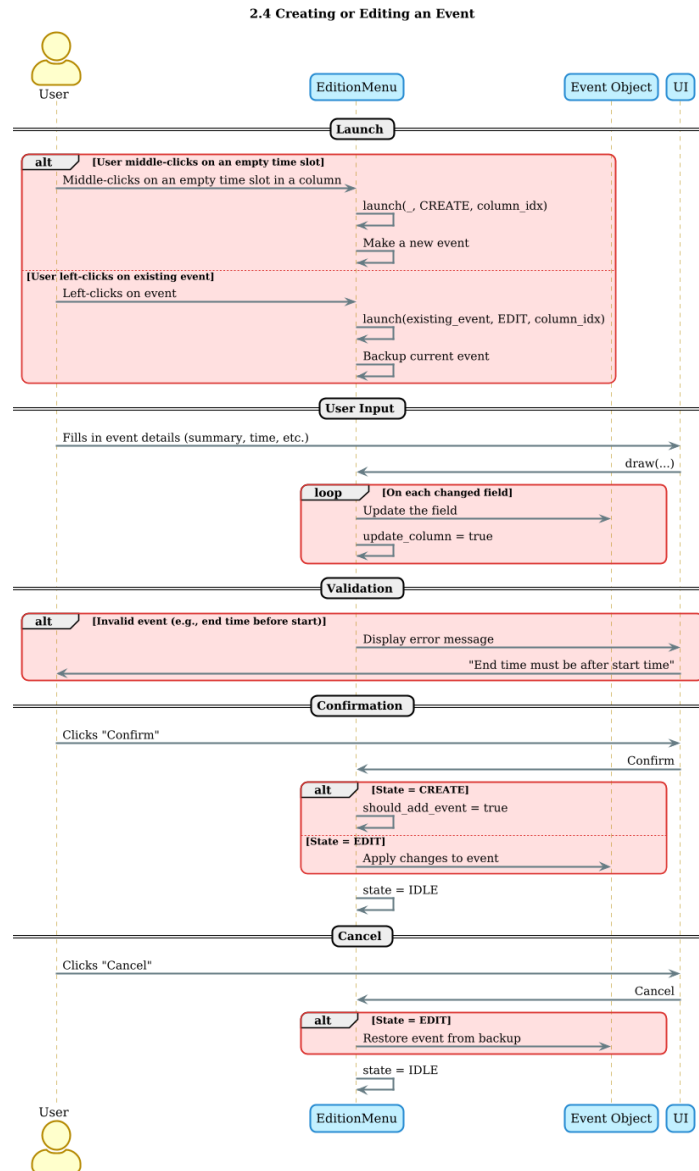
*The file was downloaded successfully.*



*The server is not responding.*

## 2.5 Creating or Editing an Event

When a user left-clicks on an existing event in the calendar, the system opens an editing window. The user can then modify the event. Once done, the user can either save or cancel the changes. If the event is invalid (e.g., the end time is earlier than the start time), the graphical interface will display an error message. When a user clicks on an empty time slot, the system will also open an editing window to add details to the new event they want to create.



**Créer un évènement**

Cours :

Fillière :

Salle :

Description :

Heure de début :  h

Heure de fin :  h

Window of creating an new Event.

**Modifier l'évènement**

Cours : PROJETS

Fillière : INFO4

Salle : PG SALLE 129 (40 pl.),PG SALLE 039 (60 pl.)

Description : INFO 4 ☐ RICHARD OLIVIER ☐ PALIX NICOLAS ☐ DC

Heure de début :  h

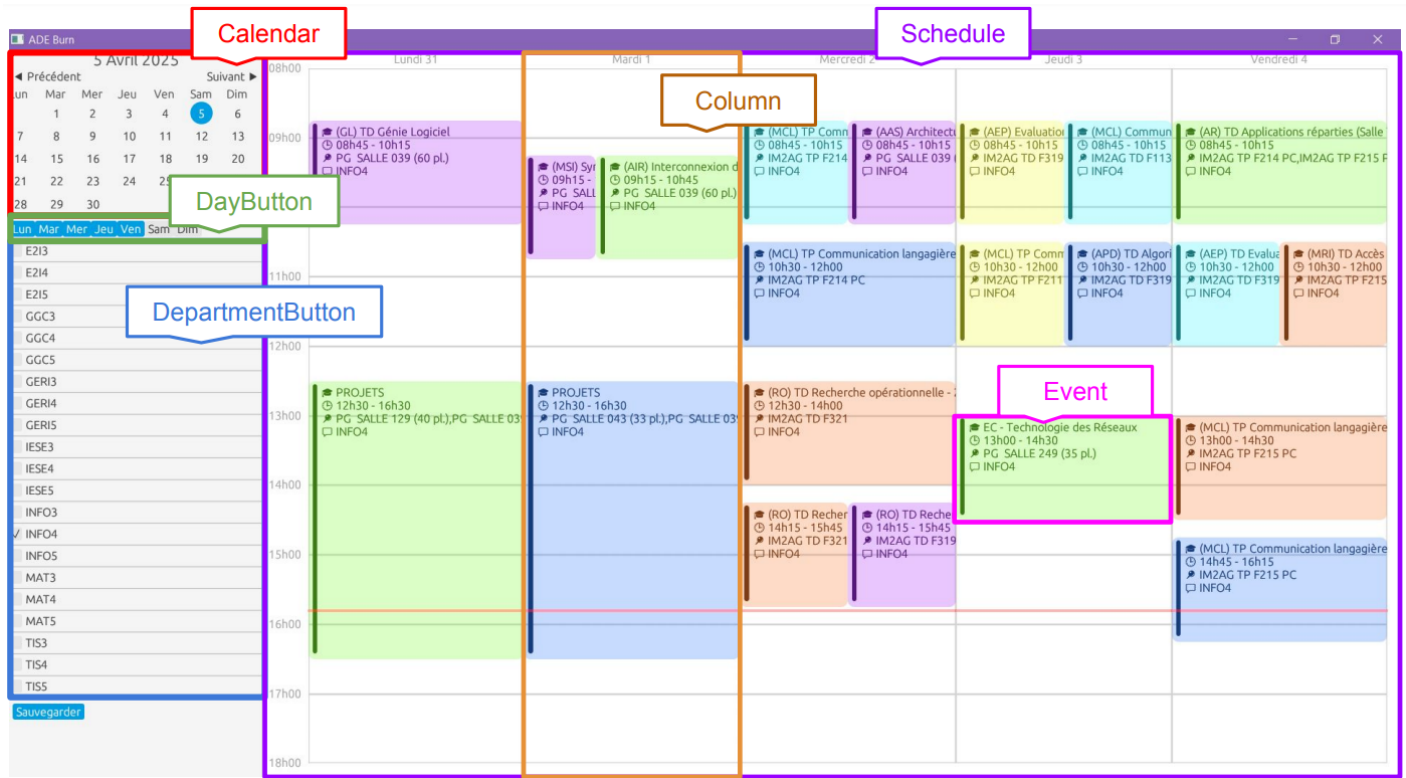
Heure de fin :  h

L'heure de fin doit être après l'heure de début

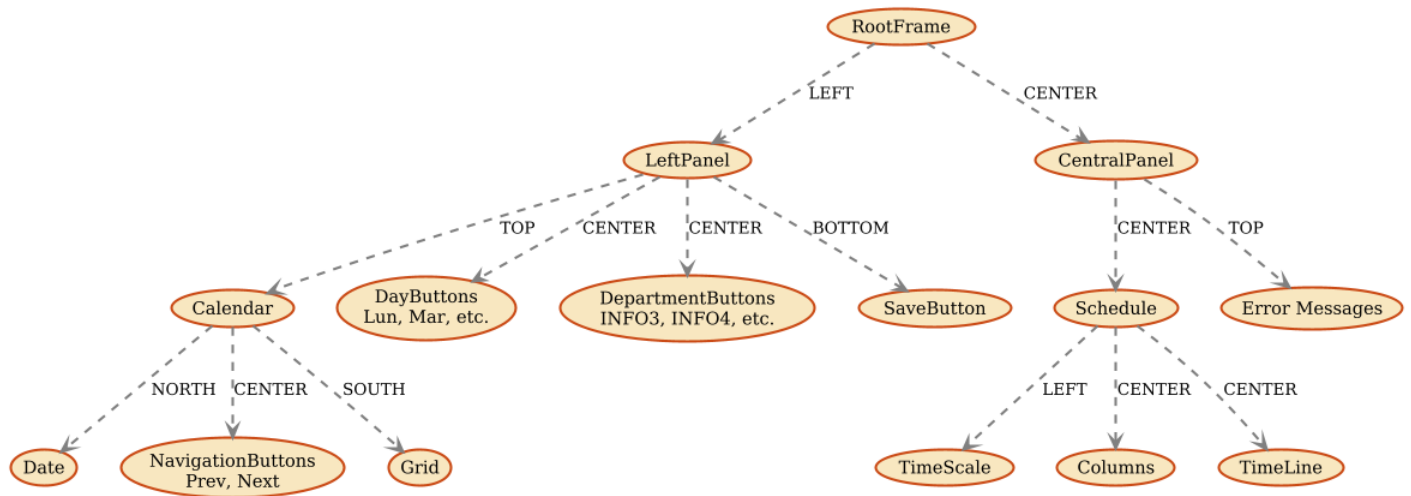
Window for editing an existing event when there is a scheduling conflict.

## 3 Structural view

### 3.1 Widgets Overview

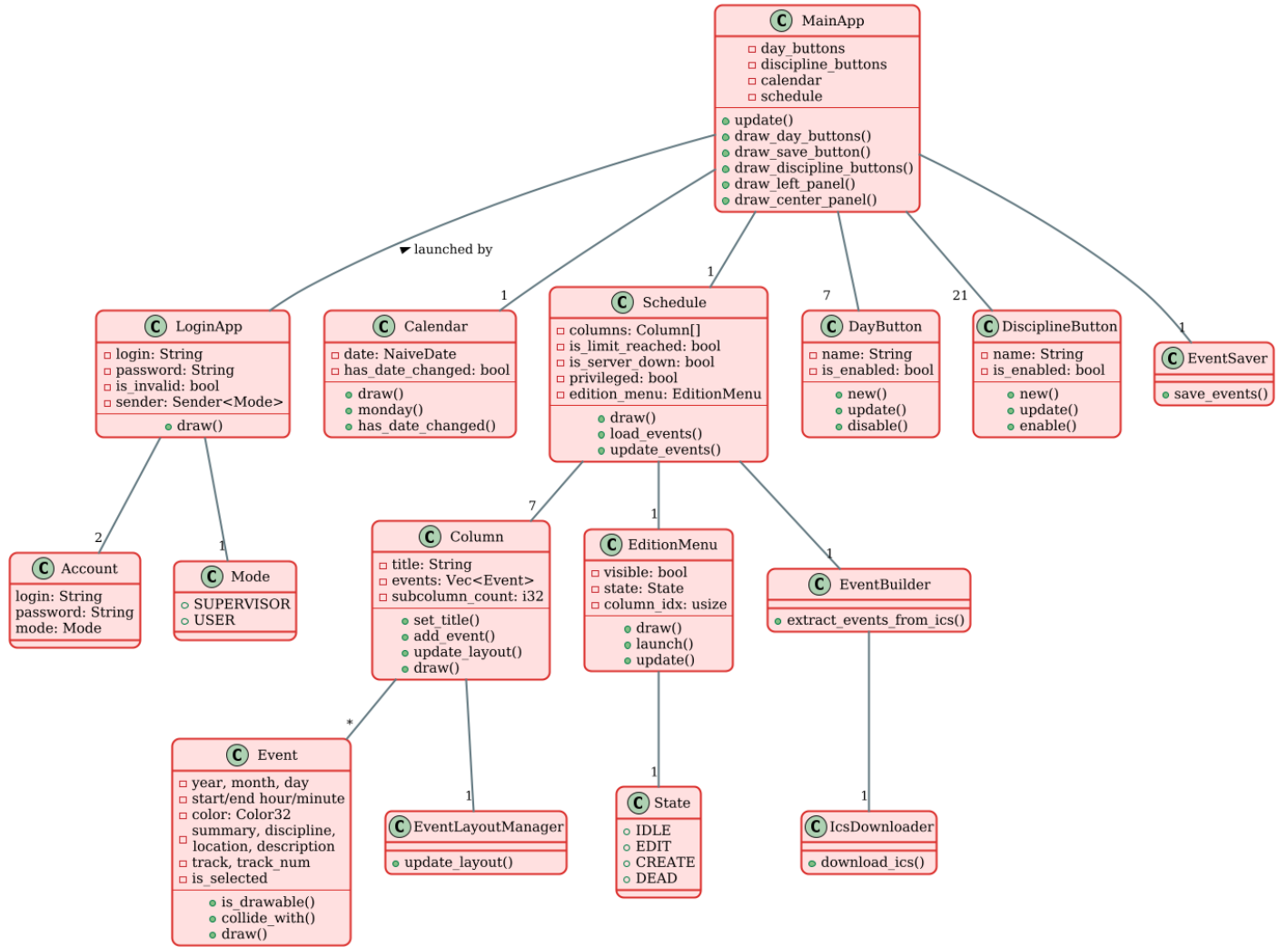


### 3.2 UI components layout





### 3.3 Links between structures



**Note :** One of the challenges with Rust is that it doesn't use objects like in Java. We tried to get as close as possible, but applying design patterns such as Singleton or using listeners wasn't straightforward. We divided the code into three parts: one for the applications (ADE and Login), one for the widgets (Event, Column, Calendar, etc.), and one for the utilities (EventSaver, EventLayoutManager, etc.), which contains the auxiliary functions.

## 4 Implementation choices

### 4.1 Event Layout Manager Algorithm

The `update_layout` algorithm assigns events to different "tracks" (columns) based on their start and end times. The algorithm follows these steps:

- **Sorting the events:** Events are first sorted by their start time, and in case of ties, by their end time.
- **Initial track assignment:** All events are initially assigned to track 0, but the first event is assigned to track 1.
- **Choosing tracks for events:** For each event, the algorithm checks if it can be assigned to an existing track without overlapping other events. If no available track is found, a new track is created, and the event is assigned to it.
- **Filling the columns:** Once the tracks are assigned, the algorithm attempts to optimize the layout by adjusting the number of tracks, ensuring that the events are placed in the minimal number of tracks required.
- **Returning the number of tracks:** Finally, the algorithm returns the total number of tracks used.



## 4.2 Suggestions for Improving the Code

Here are the things we would have liked to improve but didn't have time for:

- To improve performance and have much cleaner code: Replace `Vec<Event>` with `HashMap<u32, Event>` and use `uuid` as the key. Moreover, using a `HashMap` would prevent events from being duplicated when they are shared across multiple departments, for example with `Polyraid`, which would otherwise get duplicated.
- Replace the `String` fields for `department`, `location`, etc. with enumerations to ensure consistent data. Additionally, use a `ComboBox` instead of `TextEdit` to offer predefined choices.
- Retrieve the actual data via the ADE interface. For our tests, we manually downloaded the ICS files for each department for a month for each week.
- The application freezes when a request is sent and the response is awaited due to the `runtime.block_on` function. To resolve this, the application should be fully asynchronous.
- Since we didn't know how to implement listeners, we used a boolean variable and two functions to update a component when another one changed.
- The `event layout manager` algorithm should be revised. Events that overlap should be grouped in batches to provide a better display when expanding events to fill the column.
- For the editing feature, we could have added a history and the `ctrl-z` command to allow the administration to undo changes. In our case, we can only cancel all changes by pressing the INFO4 button for instance.  
**Note:** When pressing the save button, only the local files are updated, not those on the remote server.
- When there are scheduling conflicts (e.g., two classes assigned to the same room at the same time), the application should notify the user. This would allow them to take appropriate actions, such as modifying the schedule or reallocating resources to resolve the conflict.

## 5 Conclusion

In this project, we aimed to develop a modern, interactive timetable management application inspired by ADE, using Rust and the `egui` library. By leveraging Rust's memory safety and performance capabilities, we created a system that allows users to manage academic schedules with ease. Our application successfully incorporates dynamic timetable visualization, event creation, and real-time adjustments, while also providing a user-friendly interface for students, professors, and administrative staff.

While the application works efficiently and meets the basic requirements, there are areas for improvement. These include enhancing performance, adding more advanced features such as event duplication prevention, improving the user interface with predefined selections, and addressing potential issues with asynchronous operations. Additionally, implementing a more sophisticated event layout manager and enabling undo functionalities would significantly enhance the user experience.

This project has provided valuable insights into working with Rust and `egui`, and the experience gained in tackling challenges such as managing schedules, dealing with concurrent requests, and optimizing event layouts will serve as a solid foundation for future improvements or similar projects.

## 6 References

- You can find the source code of the project here: <https://gricad-gitlab.univ-grenoble-alpes.fr/Projets-INF04/24-25/21/ade-burn>
- Additionally, the documentation, all diagrams, and the project's evolution with the logbook are available here: <https://gricad-gitlab.univ-grenoble-alpes.fr/Projets-INF04/24-25/21/docs>