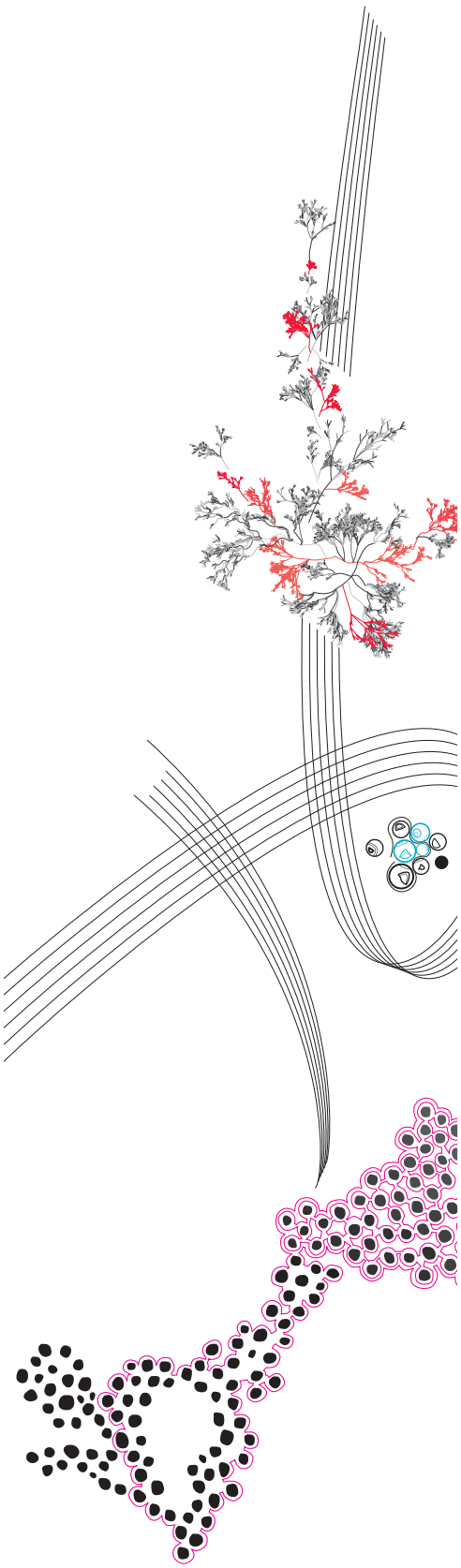


BSc Computer Science
Design Project

BMS International Relationship Database

Bart Griepsma s2988348
Dirck Mulder s2997894
Jelmer Otten s2971429
Joris Faas s2985888
Jort Bork s2980797
Thomas Brants s2920352

Supervisor: David Huistra

November, 2025

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Contents

Abstract

The BMS International Relationship Database (BIRD) is a centralized system designed to support the Faculty of Behavioural, Management and Social Sciences (BMS) and the International Office at the University of Twente in managing global academic collaborations. The system consolidates information on partner universities, cooperation types, funding, and contact tracking, while offering administrative tools to add and maintain records. A dashboard with filtering options and world map visualization provides staff with accessible insights into current and past partnerships. Integration with existing contract systems, such as Mobility Online and JOIN, is intended to reduce manual duplication of efforts and improve data consistency. By enhancing transparency, supporting strategic planning, and facilitating collaboration tracking, BIRD aims to strengthen the university's international relationships and improve administrative efficiency.

Chapter 1

Introduction

The University of Twente is a large organization with a number of international connections. These connections are universities, municipalities, companies, research institutes. They support a wide range of research activities, student exchanges, internships, or other projects. While these partnerships are valuable, the information surrounding them is often stored in different places and maintained by different individuals or departments. As a result, it can be challenging to gain a complete and clear understanding of the university's network, who is collaborating with whom, and what agreements or research topics are involved. This lack of centralized insight makes it more difficult to identify opportunities for new partnerships, track ongoing collaborations.

To address this problem, the BIRD (BMS International Relationship Database) was developed. BIRD is a web application that maps connections between people and organizations around research. The platform compiles data on collaborations between researchers, universities, and companies, showing outreach and extension within organizations due to funding and projects. By visualizing these interactions, BIRD provides an overview of the international network taking place within and beyond the University of Twente.

Chapter 2

Context Analysis

2.1 Problem Context

Researchers, universities, public institutions, and private sector partners are all part of the wide and varied network of international collaborations that the University of Twente (UT) maintains. However, data pertaining to these collaborations is frequently dispersed throughout various departments, networks, and systems. Because of this, it is challenging to compile a thorough and logical picture of UT's collaborative environment. Consequently, a number of difficulties emerge, such as:

- Information about collaborations and partnerships that is dispersed and irregularly recorded.
- Little information about who works with whom, what they work on, and what agreements or funding sources they use.

2.2 Purpose of BIRD

The BMS International Relationship Database (BIRD), a centralized platform for organizing and visualizing data on international research relationships, was created in order to address these issues. It offers an organized method for obtaining and analyzing data about partnerships both inside and outside the University of Twente. BIRD's primary goal is:

- To identify relationships and map the networks of collaboration between researchers, research groups, organizations, and projects.

2.3 Stakeholders

2.3.1 International Office Staff

The International Office at the BMS faculty supports both incoming and outgoing students and staff, coordinating exchange programmes, internships, graduation assignments, and international student support. Accurate and current information about international partnerships, host institutions, and mobility flows is essential to their work. Because the BMS International Relationship Database (BIRD) can offer a structured overview of international contacts and collaborations that are pertinent for managing agreements, advising students, and promoting mobility, the International Office is a significant stakeholder in the system. Their participation guarantees that BIRD contains information and features that facilitate useful decision-making and day-to-day international activity coordination within BMS.

2.3.2 Faculty of Behavioural, Management and Social Sciences (BMS)

The Faculty of Behavioural, Management and Social Sciences (BMS) is a key stakeholder in the BMS International Relationship Database (BIRD) as both its initiator and primary user. As a faculty about social sciences and technology, BMS relies on international collaboration to drive interdisciplinary research and societal impact. BIRD supports this mission by mapping and visualizing the global research connections, partnerships and projects of the faculty, allowing better insight into collaboration patterns, funding links, and research influence. By providing data, expertise, and strategic direction, BMS ensures that BIRD aligns with its goal of fostering responsible, data-driven, and globally engaged research within the University of Twente.

2.3.3 Maintainer

The system maintainer is in charge of the BMS International Relationship Database's (BIRD) ongoing development, dependability, and technical maintenance. Maintaining system stability, applying updates, controlling user accounts and permissions, and combining information from multiple internal and external sources are all part of this job. In addition, the system maintainer fixes problems, offers users technical support, and works to enhance the platform in response to stakeholder input. The system maintainer plays a crucial role in enabling the platform's functionality by making sure that BIRD continues to be safe, easy to use, and scalable to accommodate the changing requirements of the BMS faculty and its global collaborations.

Chapter 3

Planning & Approach

The project was organized around a weekly sprints to make the workload manageable and ensure consistent progress. The timeline helped the team stay focused on the short-term goals while keeping the bigger picture in mind. Each week had a specific theme or set of deliverables, which made it easier to track progress and discuss updates during supervision meetings.

The first week was spent setting up the environment and tools needed for development. We created the GitLab repository, configured a runner for continuous integration, and set up Notion and Jira for task tracking and documentation. These tools helped us to clearly divide tasks between team members and visualize the workflow. We also had an initial meeting with the supervisor to discuss the scope, expectations, and technical requirements of the project.

Once the setup phase was completed, we moved on to the development work. The backend team started by designing the database schema and setting up Django models and APIs. At the same time, the frontend team initialized the Nuxt project and built the first version of the user interface. From then on, most of the weeks focused on adding features, improving functionality, and integrating the frontend and backend parts of the system.

During weeks 3-8 of the project, we implemented key features such as authentication, form validation, and contact management. Integration between the two parts of the system was done continuously so that issues could be resolved early. This also allowed the frontend to stay in sync with updates on the backend, such as changes in the database design or API endpoints.

In week 8-10, the focus shifted towards polishing and deployment. We worked on improving security, setting up the Docker configuration and deploying the application to the university's Kubernetes cluster. Some of the original planning had to be adjusted when new security restrictions were introduced by the university, which prevented containers from writing data internally. To solve this, we mounted external volumes to make data storage

persistent. These kinds of issues required some flexibility in the schedule, but we managed to stay close to the original timeline.

The final stage of the project focused on testing, documentation, and preparing the final report and delivery. By that point, most of the core functionality was complete, so the remaining time was used to fix bugs, refine the user experience, and write supporting documentation for the dashboard.

A complete overview of the planning, including all tasks and milestones, is shown in Appendix B., which contains the full Gantt chart of the project.

Chapter 4

Interviews

This project involved ongoing consultation with the supervisor and the International Office to define requirements, review design prototypes, and ensure alignment with existing processes.

4.1 Meeting – 8 September

Main Topics: Relations between fundings, networks, institutions, and groups; data structure; permissions.

Summary: The discussion focused on understanding how fundings and cooperations relate to networks, institutions, faculties, and groups. It was emphasized that contact moments should be linked not only to individual researchers but also to larger entities such as research groups, faculties, or institutions. The team discussed the need for a system that maintains a complete archive of interactions, including informal contacts from conferences. Permissions were addressed, distinguishing between full access for International Office staff and limited rights for researchers

Key points:

- Contact moments may be linked to individuals, networks, or cooperations.
- Exporting data to Excel should be limited to essential purposes.
- System should maintain an archive of meetings and contact moments, including informal interactions at conferences.
- Cooperations may include multiple institutions; networks may include individual researchers or cooperations.

- Clear labeling of persons, cooperations, and networks is essential.
- Permissions: International Office staff require full access; researchers need limited access. Confidential information should be visible only to authorized staff.

4.2 Meeting – 24 September

Main Topics: Frontend and backend design; data workflows; dashboards; gamification; user engagement.

Summary: The team reviewed initial frontend designs and backend structures. It was decided that contact moments should be sortable and filterable by person, institution, cooperation, or network. Institutions were prioritized in visualizations, and dashboards were proposed to show links between universities. Labels were suggested to indicate faculties, research groups, and study programs. Ideas for gamifying the system to encourage engagement were introduced, and alignment with the UT website design was discussed.

Key points:

- Contact moments should be sortable/filterable by person, institution, cooperation, or network.
- Institutions should be prioritized over individuals in visual representation.
- Dashboard should highlight connections between universities and display labels such as faculty, research group, and study program.
- Integration of exchange students and cooperation agreements.
- Gamification ideas to encourage frequent usage, showing user activity metrics.
- Design consistency with the UT website.

4.3 Meeting – 26 September

Main Topics: Data management workflows; adding institutions, cooperations, and individuals; labeling and filtering.

Summary: Data entry and workflow processes were streamlined. All connections were to reflect the UT perspective. Group hierarchies were replaced with a labeling system for faculties, study programs, and research groups. Dashboard filtering features were confirmed, allowing users to view institutions, cooperation types, and connections. Permissions

were finalized to restrict edits to creators and administrators.

Key points:

- Streamlined procedures for adding institutions, cooperations, and persons.
- Connections between persons and cooperations must always reflect the UT perspective.
- Dashboard features for filtering institutions, types of cooperations, and connections.
- Group hierarchies replaced with labels for faculties, study programs, and research groups.
- Permissions: only creators or administrators can edit entries.

4.4 Meeting – 2 October

Main Topics: Technical implementation; Django polymorphic models; exchange student data.

Summary: Implementation details were discussed, confirming the use of polymorphic Django models for cooperations, networks, and funding types. The system was updated to display exchange students and pre-fill commonly used fields to streamline workflow.

Key points:

- Polymorphic Django data model confirmed for managing entities (cooperations, networks, fundings).
- Display exchange students in the system.
- Pre-fill relevant form fields to improve data entry efficiency.

4.5 Meeting – 9 October

Main Topics: User workflows; timeline creation; privacy; integration with other systems.

Summary: The "happy flow" for daily tasks was reviewed, including automatic date population and pre-filled fields. Timelines for contact moments and cooperative agreements were introduced. Dropdown selections were optimized, address validation for mapping features was added, and privacy controls were discussed.

Key points:

- Default date to current day and pre-fill common fields.
- Timeline views for contact moments and cooperative agreements.
- Plan future meetings for integrating additional systems (e.g., BIRD).
- Interface improvements: dropdown optimization, address validation, privacy features.

4.6 Interview – JOIN and Mobility Online

Main Topics: Integration with external systems (Mobility Online, JOIN); labeling; permissions.

Summary: The interview focused on linking data from Mobility Online and JOIN to the dashboard. The International Office currently uses Excel exports to track exchange agreements, institutions, and contacts. Insights were gathered about labeling and filtering by faculty, research group, and study level. Permissions were confirmed, ensuring full access for International Office staff and limited access for researchers.

Key points:

- Mobility Online: handles exchange agreements; JOIN: handles cooperation agreements.
- Current use of Excel exports for internal/external reporting.
- Consolidate labels for faculties, study programs, and PhD tracks.
- Historical contact moments should be visible and linked to ongoing agreements.
- Permissions: IO staff full access; researchers limited access.
- Integration points identified for a unified dashboard interface.

4.7 Meeting – 23 October

Main Topics: Dashboard finalization; permissions; user workflows; interface adjustments.

Summary: The dashboard was finalized as the central hub for institutions, networks, and cooperations. Permissions and privacy settings were confirmed, and user workflows were validated. Interface adjustments were reviewed to ensure smooth and consistent experience for all users.

Key points:

- Dashboard landing page as central hub for institutions, networks, and cooperations.
- Finalize permissions and privacy controls.
- Validate user workflows and finalize remaining interface adjustments.

Chapter 5

User Stories and Requirements

5.1 Functional Requirements

The functional requirements highlight what the system must be able to do, in order to achieve the user needs. They describe the actions and outcomes that stakeholders expect when interacting with the different parts of the dashboard. In the following subsections, we explain further what the required functionalities are for each entity within the system. We assign the right urgency to these requirements in order to get a better picture of what the most important features are.

5.1.1 Authentication and Authorization

- **FR1 (Must):** The system must allow users to authenticate using Microsoft accounts via OAuth 2.0.
- **FR2 (Must):** The system must require authentication for all API endpoints except login.
- **FR3 (Must):** The system must issue JWT tokens upon successful authentication for API access.
- **FR4 (Must):** The system must support token refresh to maintain user sessions.
- **FR5 (Must):** The system must redirect authenticated users away from the login page.
- **FR6 (Must):** The system must redirect unauthenticated users to the login page with a return URL.

- **FR7 (Should):** The system should automatically create user accounts upon first Microsoft login.
- **FR8 (Must):** The system must support user roles with staff and regular user distinctions.

5.1.2 Institution Management

- **FR9 (Must):** The system must allow users to create, read, update, and delete institution records.
- **FR10 (Must):** The system must require institution name and country as mandatory fields.
- **FR11 (Must):** The system must allow institutions to be categorized by type.
- **FR12 (Must):** The system must support associating multiple persons with an institution.
- **FR13 (Must):** The system must support assigning multiple labels to institutions.
- **FR14 (Must):** The system must allow institutions to have addresses with geocoding capabilities.
- **FR15 (Must):** The system must display institutions on an interactive map.
- **FR16 (Should):** The system should automatically geocode addresses when created or updated.
- **FR17 (Must):** The system must allow searching institutions by name.
- **FR18 (Must):** The system must allow filtering institutions by type, country, and labels.
- **FR19 (Must):** The system must display all cooperations associated with an institution.

5.1.3 Person Management

- **FR20 (Must):** The system must allow users to create, read, update, and delete person records.
- **FR21 (Must):** The system must require first name and last name as mandatory fields.

- **FR23 (Must):** The system must allow creating connections between persons and institutions.
- **FR24 (Must):** The system must allow creating connections between persons.
- **FR26 (Must):** The system must prevent duplicate person-person connections.
- **FR27 (Must):** The system must prevent persons from being connected to themselves.
- **FR28 (Must):** The system must allow assigning labels to person-institution connections.
- **FR29 (Must):** The system must allow assigning labels to person-person connections.
- **FR30 (Must):** The system must allow searching persons by name or email.
- **FR31 (Must):** The system must allow filtering persons by associated institutions.
- **FR32 (Must):** The system must display all institutions and person connections for a person.

5.1.4 Agreement Management

- **FR33 (Must):** The system must allow users to create, read, update, and delete agreement records.
- **FR34 (Must):** The system must require start date and end date for agreements.
- **FR35 (Must):** The system must allow agreements to be categorized by type.
- **FR36 (Must):** The system must support associating multiple institutions with an agreement.
- **FR37 (Must):** The system must support assigning multiple labels to agreements.
- **FR38 (Must):** The system must allow marking agreements as staff-only.
- **FR39 (Must):** The system must restrict staff-only agreements from regular users.
- **FR40 (Must):** The system must allow searching agreements by name.
- **FR41 (Must):** The system must allow filtering agreements by type, date range, institutions, and labels.
- **FR42 (Should):** The system should allow adding comments to agreements.
- **FR43 (Must):** The system must track who created and updated each agreement.

5.1.5 Network Management

- **FR44 (Must):** The system must allow users to create, read, update, and delete network records.
- **FR45 (Must):** The system must require a name for networks.
- **FR46 (Must):** The system must support associating multiple institutions with a network.
- **FR47 (Must):** The system must support assigning multiple labels to networks.
- **FR48 (Must):** The system must allow marking networks as staff-only.
- **FR49 (Must):** The system must restrict staff-only networks from regular users.
- **FR50 (Must):** The system must allow searching networks by name.
- **FR51 (Must):** The system must allow filtering networks by associated institutions and labels.
- **FR52 (Must):** The system must track who created and updated each network.

5.1.6 Funding Management

- **FR53 (Must):** The system must allow users to create, read, update, and delete funding records.
- **FR54 (Must):** The system must require a name, start date, and end date for fundings.
- **FR55 (Must):** The system must allow fundings to be categorized by type.
- **FR56 (Must):** The system must support associating multiple institutions with a funding.
- **FR57 (Must):** The system must support assigning multiple labels to fundings.
- **FR58 (Must):** The system must allow marking fundings as staff-only.
- **FR59 (Must):** The system must restrict staff-only fundings from regular users.
- **FR60 (Must):** The system must allow searching fundings by name.
- **FR61 (Must):** The system must allow filtering fundings by type, date range, institutions, and labels.
- **FR62 (Must):** The system must track who created and updated each funding.

5.1.7 Contact Moment Management

- **FR63 (Must):** The system must allow users to create, read, update, and delete contact moment records.
- **FR64 (Must):** The system must require a description for contact moments.
- **FR65 (Must):** The system must allow contact moments to be categorized by type with color coding.
- **FR66 (Must):** The system must support associating multiple person-institution connections with contact moments.
- **FR67 (Must):** The system must support assigning multiple labels to contact moments.
- **FR68 (Must):** The system must allow marking contact moments as staff-only.
- **FR69 (Must):** The system must restrict staff-only contact moments from regular users.
- **FR70 (Must):** The system must allow searching contact moments by description.
- **FR71 (Must):** The system must allow filtering contact moments by type, date, associated persons/institutions, and labels.
- **FR72 (Should):** The system should allow adding comments to contact moments.
- **FR73 (Must):** The system must track who created and updated each contact moment.

5.1.8 Label Management

- **FR74 (Must):** The system must allow users to create, read, update, and delete label records.
- **FR75 (Must):** The system must require category and text fields for labels.
- **FR76 (Must):** The system must allow searching labels by text or category.
- **FR77 (Must):** The system must allow filtering labels by category.
- **FR78 (Must):** The system must support assigning labels to institutions, persons, agreements, networks, fundings, and contact moments.
- **FR79 (Must):** The system must track who created and updated each label.

5.1.9 Comment Management

- **FR80 (Should):** The system should allow users to add comments to agreements.
- **FR81 (Should):** The system should allow users to add comments to contact moments.
- **FR82 (Should):** The system should display comments in chronological order.
- **FR83 (Should):** The system should track who created and updated each comment.

5.1.10 Timeline and Audit Trail

- **FR84 (Should):** The system should track all create, update, and delete operations.
- **FR85 (Should):** The system should record which user performed each operation.
- **FR86 (Should):** The system should record timestamps for all operations.
- **FR87 (Should):** The system should store old and new values for updated fields.
- **FR88 (Should):** The system should display timeline events in reverse chronological order.
- **FR89 (Should):** The system should allow viewing timeline events on the dashboard.

5.1.11 Data Display and Navigation

- **FR90 (Must):** The system must provide a dashboard showing institutions map, timeline, and quick access.
- **FR91 (Must):** The system must allow sorting data tables by any column.
- **FR92 (Must):** The system must support pagination for large data sets.
- **FR93 (Must):** The system must allow global search across relevant fields.
- **FR94 (Should):** The system should allow showing and hiding columns in data tables.
- **FR95 (Should):** The system should allow selecting multiple items for bulk operations.
- **FR96 (Should):** The system should display item counts in filtered lists.
- **FR97 (Should):** The system should allow clearing all filters at once.

5.1.12 Settings Management

- **FR98 (Should):** The system should allow managing institution types.
- **FR99 (Should):** The system should allow managing agreement types.
- **FR100 (Should):** The system should allow managing funding types.
- **FR101 (Should):** The system should allow managing contact moment types with color assignment.
- **FR102 (Should):** The system should allow bulk deletion of types and labels.

5.1.13 Geographic Features

- **FR103 (Should):** The system should display institutions on an interactive map using Leaflet.
- **FR104 (Should):** The system should show institution markers with popup information.
- **FR105 (Should):** The system should only display institutions with valid geocoded addresses on the map.
- **FR106 (Could):** The system could provide map clustering for institutions in close proximity.

5.2 Non-Functional Requirements

5.2.1 Security

- **NFR1 (Must):** The system must enforce HTTPS in production environments.
- **NFR2 (Must):** The system must use secure session cookies in production.
- **NFR3 (Must):** The system must implement CSRF protection for all state-changing operations.
- **NFR4 (Must):** The system must sanitize user input to prevent XSS attacks using django-bleach.
- **NFR5 (Must):** The system must restrict API access to authenticated users only.
- **NFR6 (Must):** The system must enforce staff-only content restrictions at the API level.

- **NFR7 (Must):** The system must validate all input data before processing.
- **NFR8 (Should):** The system should implement secure password validation rules.
- **NFR9 (Could):** The system should log security-related events for audit purposes.

5.2.2 Performance

- **NFR10 (Should):** The system should optimize database queries using prefetch.
- **NFR12 (Should):** The system should paginate API responses to limit response sizes.
- **NFR13 (Should):** The system should serve static files efficiently.
- **NFR14 (Could):** The system could implement database query result caching for frequently accessed data.
- **NFR15 (Could):** The system could optimize map rendering for large numbers of institutions.

5.2.3 Usability

- **NFR16 (Must):** The system must provide clear error messages for user actions.
- **NFR17 (Must):** The system must provide loading indicators for asynchronous operations.
- **NFR18 (Should):** The system should provide responsive design for different screen sizes.
- **NFR19 (Should):** The system should provide keyboard navigation support.
- **NFR20 (Should):** The system should provide accessible form labels.
- **NFR21 (Should):** The system should provide consistent navigation and layout patterns.

5.2.4 Data Integrity

- **NFR23 (Must):** The system must enforce unique constraints on email addresses for persons.
- **NFR24 (Must):** The system must enforce unique constraints on person-institution connections.

- **NFR25 (Must):** The system must enforce unique constraints on person-person connections.
- **NFR26 (Must):** The system must prevent self-connections for persons.
- **NFR27 (Must):** The system must maintain referential integrity for foreign key relationships.
- **NFR28 (Must):** The system must validate date ranges for agreements and fundings.

5.2.5 Reliability

- **NFR30 (Must):** The system must handle database connection failures gracefully.
- **NFR32 (Should):** The system should provide error recovery mechanisms for failed operations.
- **NFR33 (Should):** The system should log errors for debugging and monitoring.

5.2.6 Maintainability

- **NFR38 (Must):** The system must follow Django REST Framework conventions for API structure.
- **NFR39 (Should):** The system should provide comprehensive code documentation.
- **NFR40 (Should):** The system should use consistent naming conventions across the codebase.

5.3 System Constraints

- **C1:** The backend must be implemented using Django and Django REST Framework
- **C5:** The system must use SQLite for development and must support PostgreSQL for production.
- **C6:** The system must use JWT tokens for API authentication via djangorestframework-simplejwt.
- **C7:** The system must integrate with Microsoft Azure AD for user authentication via django-allauth.
- **C13:** The system must be deployed using Docker containers.
- **C15:** The system must be hosted on University of Twente infrastructure with domain *.utwente.nl.

5.4 User stories

Now that we have established the requirements for the system, we will elaborate on these requirements further by mentioning the specific user stories. The user stories describe the active action that the user wants to achieve, and why they want to achieve this. These are again divided into the correct sub-chapters according to the entities of the system.

5.4.1 Authentication

- As a user, I want to sign in with my Microsoft account, so that I can access the system securely.
- As a user, I want to be redirected to the dashboard after successful login, so that I can start using the system immediately.

5.4.2 Dashboard

- As a user, I want to view a map showing all institutions, so that I can see their geographical distribution.
- As a user, I want to see a timeline of recent system changes, so that I can track what has been modified.
- As a user, I want to see detailed change information in the timeline, so that I can understand what fields were modified.
- As a user, I want quick access to institutions from the dashboard, so that I can navigate to them efficiently.
- As a user, I want to see the number of networks and persons associated with each institution on the dashboard, so that I can quickly assess their importance.

5.4.3 Institutions

- As a user, I want to view a list of all institutions, so that I can browse the partner database.
- As a user, I want to search institutions by name, so that I can quickly find specific partners.
- As a user, I want to filter institutions by type, so that I can focus on specific categories.
- As a user, I want to filter institutions by country, so that I can see partners in specific regions.

- As a user, I want to filter institutions by labels, so that I can find institutions with specific tags.
- As a user, I want to view detailed information about an institution, so that I can see all its associated data.
- As a user, I want to create a new institution, so that I can add new partners to the database.
- As a user, I want to edit institution information, so that I can keep the database up to date.
- As a user, I want to delete an institution, so that I can remove obsolete entries.
- As a user, I want to see the address of an institution automatically geocoded on a map, so that I can visualize its location.
- As a user, I want to associate persons with institutions, so that I can track who works at which partner organization.
- As a user, I want to assign labels to institutions, so that I can categorize them for better organization.
- As a user, I want to see all cooperations associated with an institution, so that I can understand the relationship.

5.4.4 Persons

- As a user, I want to view a list of all persons so that I can see the details of all contacts.
- As a user, I want to search persons by name, so that I can quickly find specific contacts.
- As a user, I want to filter persons by associated institutions, so that I can see contacts from specific organizations.
- As a user, I want to view detailed information about a person, so that I can see all their connections and relationships.
- As a user, I want to create a new person, so that I can add new contacts to the database.
- As a user, I want to edit person information, so that I can update contact details.
- As a user, I want to delete a person, so that I can remove obsolete contacts.

- As a user, I want to create connections between persons, so that I can track relationships.
- As a user, I want to assign labels to person connections, so that I can categorize the type of relationship.
- As a user, I want to see all institutions a person is associated with, so that I can understand their affiliations.
- As a user, I want to see all person-to-person connections, so that I can understand the network of relationships.

5.4.5 Agreements

- As a user, I want to view a list of all agreements, so that I can see all active and past partnerships.
- As a user, I want to search agreements by name, so that I can quickly find specific agreements.
- As a user, I want to filter agreements by type, so that I can focus on specific agreement categories.
- As a user, I want to filter agreements by date range, so that I can see agreements active during specific periods.
- As a user, I want to filter agreements by associated institutions, so that I can see all agreements with specific partners.
- As a user, I want to filter agreements by labels, so that I can find agreements with specific tags.
- As a user, I want to view detailed information about an agreement, so that I can see all its terms and associations.
- As a user, I want to create a new agreement, so that I can record new partnerships.
- As a user, I want to edit agreement information, so that I can update terms and details.
- As a user, I want to delete an agreement, so that I can remove obsolete records.
- As a user, I want to specify start and end dates for agreements, so that I can track their duration.
- As a user, I want to associate multiple institutions with an agreement, so that I can track multi-party partnerships.

- As a user, I want to assign labels to agreements, so that I can categorize them for better organization.
- As a user, I want to add comments to agreements, so that I can document important notes and discussions.
- As a user, I want to mark agreements as staff-only, so that I can restrict sensitive information to authorized personnel.

5.4.6 Networks

- As a user, I want to view a list of all networks, so that I can see all research networks.
- As a user, I want to search networks by name, so that I can quickly find specific networks.
- As a user, I want to filter networks by associated institutions, so that I can see networks involving specific partners.
- As a user, I want to filter networks by labels, so that I can find networks with specific tags.
- As a user, I want to view detailed information about a network, so that I can see all its members and details.
- As a user, I want to create a new network, so that I can record new research collaborations.
- As a user, I want to edit network information, so that I can update network details.
- As a user, I want to delete a network, so that I can remove obsolete records.
- As a user, I want to associate multiple institutions with a network, so that I can track all participating organizations.
- As a user, I want to assign labels to networks, so that I can categorize them for better organization.
- As a user, I want to mark networks as staff-only, so that I can restrict sensitive information to authorized personnel.

5.4.7 Fundings

- As a user, I want to view a list of all fundings, so that I can see all funding opportunities and grants.
- As a user, I want to search fundings by name, so that I can quickly find specific funding programs.
- As a user, I want to filter fundings by type, so that I can focus on specific funding categories.
- As a user, I want to filter fundings by date range, so that I can see fundings active during specific periods.
- As a user, I want to filter fundings by associated institutions, so that I can see fundings involving specific partners.
- As a user, I want to filter fundings by labels, so that I can find fundings with specific tags.
- As a user, I want to view detailed information about a funding, so that I can see all its details and associations.
- As a user, I want to create a new funding, so that I can record new funding opportunities.
- As a user, I want to edit funding information, so that I can update funding details.
- As a user, I want to delete a funding, so that I can remove obsolete records.
- As a user, I want to specify start and end dates for fundings, so that I can track their duration.
- As a user, I want to associate multiple institutions with a funding, so that I can track all participating organizations.
- As a user, I want to assign labels to fundings, so that I can categorize them for better organization.
- As a user, I want to mark fundings as staff-only, so that I can restrict sensitive information to authorized personnel.

5.4.8 Contact Moments

- As a user, I want to view a list of all contact moments, so that I can see all interactions with partners.
- As a user, I want to search contact moments by description, so that I can quickly find specific interactions.
- As a user, I want to filter contact moments by type, so that I can focus on specific interaction categories.
- As a user, I want to filter contact moments by date, so that I can see interactions during specific periods.
- As a user, I want to filter contact moments by associated persons and institutions, so that I can see interactions with specific contacts.
- As a user, I want to filter contact moments by labels, so that I can find interactions with specific tags.
- As a user, I want to view detailed information about a contact moment, so that I can see all its details and associations.
- As a user, I want to create a new contact moment, so that I can record interactions with partners.
- As a user, I want to edit contact moment information, so that I can update interaction details.
- As a user, I want to delete a contact moment, so that I can remove obsolete records.
- As a user, I want to specify a date for contact moments, so that I can track when interactions occurred.
- As a user, I want to specify a location for contact moments, so that I can record where interactions took place.
- As a user, I want to associate persons and institutions with contact moments, so that I can track who was involved in the interaction.
- As a user, I want to assign labels to contact moments, so that I can categorize them for better organization.
- As a user, I want to add comments to contact moments, so that I can document additional details and follow-ups.
- As a user, I want to mark contact moments as staff-only, so that I can restrict sensitive information to authorized personnel.

5.4.9 Labels

- As a user, I want to view all labels, so that I can see available categorization options.
- As a user, I want to search labels by text or category, so that I can quickly find specific labels.
- As a user, I want to filter labels by category, so that I can see labels in specific groups.
- As a user, I want to create a new label, so that I can add new categorization options.
- As a user, I want to edit label information, so that I can update category and text.
- As a user, I want to delete labels, so that I can remove obsolete categorization options.
- As a user, I want to assign labels to institutions, so that I can categorize partners.
- As a user, I want to assign labels to persons, so that I can categorize contacts.
- As a user, I want to assign labels to agreements, so that I can categorize partnerships.
- As a user, I want to assign labels to networks, so that I can categorize research collaborations.
- As a user, I want to assign labels to fundings, so that I can categorize funding opportunities.
- As a user, I want to assign labels to contact moments, so that I can categorize interactions.

5.4.10 Comments

- As a user, I want to view all comments on an agreement, so that I can see all notes and discussions.
- As a user, I want to view all comments on a contact moment, so that I can see all additional details.
- As a user, I want to add a comment to an agreement, so that I can document important notes.
- As a user, I want to add a comment to a contact moment, so that I can document follow-up actions.
- As a user, I want to see who created each comment and when, so that I can track the source of information.
- As a user, I want to see when comments were last updated, so that I can track recent changes.

5.4.11 Timeline

- As a user, I want to view a timeline of all system changes, so that I can track modifications across the system.
- As a user, I want to see what entity was changed in the timeline, so that I can identify what was modified.
- As a user, I want to see what action was performed (created, updated, deleted), so that I can understand the type of change.
- As a user, I want to see who made the change, so that I can track who is responsible for modifications.
- As a user, I want to see when the change was made, so that I can understand the timeline of events.
- As a user, I want to see what fields were changed, so that I can understand the scope of modifications.
- As a user, I want to see the old and new values of changed fields, so that I can understand what was modified.

5.4.12 Settings

- As a user, I want to view all institution types, so that I can see available categories.
- As a user, I want to create institution types, so that I can add new institution categories.
- As a user, I want to edit institution types, so that I can update category names.
- As a user, I want to delete institution types, so that I can remove obsolete categories.
- As a user, I want to view all agreement types, so that I can see available agreement categories.
- As a user, I want to create agreement types, so that I can add new agreement categories.
- As a user, I want to edit agreement types, so that I can update category names.
- As a user, I want to delete agreement types, so that I can remove obsolete categories.
- As a user, I want to view all funding types, so that I can see available funding categories.

- As a user, I want to create funding types, so that I can add new funding categories.
- As a user, I want to edit funding types, so that I can update category names.
- As a user, I want to delete funding types, so that I can remove obsolete categories.
- As a user, I want to view all contact moment types, so that I can see available interaction categories.
- As a user, I want to create contact moment types, so that I can add new interaction categories.
- As a user, I want to edit contact moment types, so that I can update category names and colors.
- As a user, I want to delete contact moment types, so that I can remove obsolete categories.
- As a user, I want to assign colors to contact moment types, so that I can visually distinguish different interaction types.

5.4.13 Data Management

- As a user, I want to sort data tables by different columns, so that I can organize information according to my needs.
- As a user, I want to paginate through large lists of data, so that I can navigate efficiently.
- As a user, I want to select multiple items in lists, so that I can perform bulk operations.
- As a user, I want to show or hide columns in data tables, so that I can customize the view.
- As a user, I want to clear all filters at once, so that I can quickly reset my view.
- As a user, I want to see the total count of items in filtered lists, so that I can understand the scope of results.

5.4.14 Geographic Features

- As a user, I want to see institutions displayed on a map, so that I can visualize their geographic distribution.
- As a user, I want to see institution markers on the map, so that I can identify their locations.

- As a user, I want to click on map markers to see institution details, so that I can quickly access information.
- As a user, I want addresses to be automatically geocoded when entered, so that I can see institutions on the map without manual coordinate entry.
- As a user, I want to see only institutions with valid addresses on the map, so that I can focus on geocoded locations.

5.4.15 Staff-Only Content

- As a staff member, I want to mark agreements as staff-only, so that I can restrict sensitive partnership information.
- As a staff member, I want to mark networks as staff-only, so that I can restrict sensitive collaboration information.
- As a staff member, I want to mark fundings as staff-only, so that I can restrict sensitive funding information.
- As a staff member, I want to mark contact moments as staff-only, so that I can restrict sensitive interaction information.
- As a staff member, I want to view staff-only content, so that I can access all system information.
- As a regular user, I want to be prevented from viewing staff-only content, so that sensitive information remains protected.

5.4.16 Address Management

- As a user, I want to enter addresses for institutions, so that I can record their physical locations.
- As a user, I want to specify only a country for an address, so that I can record minimal location information.
- As a user, I want to see addresses automatically geocoded with coordinates, so that they appear on the map.
- As a user, I want addresses to be updated when I modify address fields, so that coordinates stay current.

5.4.17 Person-Institution Connections

- As a user, I want to create connections between persons and institutions, so that I can track who works at which organization.
- As a user, I want to assign labels to person-institution connections, so that I can categorize the type of relationship (e.g., employee, consultant, board member).
- As a user, I want to see all persons associated with an institution, so that I can identify key contacts.
- As a user, I want to see all institutions associated with a person, so that I can understand their affiliations.
- As a user, I want to prevent duplicate person-institution connections, so that data integrity is maintained.

5.4.18 Person-Person Connections

- As a user, I want to create connections between persons, so that I can track professional relationships.
- As a user, I want to assign labels to person-person connections, so that I can categorize the type of relationship.
- As a user, I want to see all connections for a person, so that I can understand their network.
- As a user, I want to prevent duplicate person-person connections, so that data integrity is maintained.
- As a user, I want to prevent persons from being connected to themselves, so that data quality is maintained.

Chapter 6

User Interface Design

6.1 Design Rationale and Approach

6.1.1 Overall Design Goals

For the design of the BIRD system, accessability and usability were identified as the two main design principles. Since BIRD acts as a central platform for managing a wide range of relationships with other institutions, the interface needed to facilitate both an increasing number of relationships, and a diverse group of users across the University of Twente.

Accessibility refers not only to the system's technical capacity to handle growing data sets, but also to the flexibility of its interface to display and manage expanding networks of persons, institutions, agreements, and collaborations. Users should be able to quickly locate the right data, explore relationships between entities, and add new records without unnecessary complexity. To achieve this, the interface was designed around clear information hierarchies and simple navigation patterns that reduce the number of steps needed within the sytem. Thus making it easier to navigate and work with.

Usability emphasises that the system should be usable by a broad range of staff members within the university, not only those who interact with it daily. Also the users that do not interact with it daily, should be able to quickly understand the interface. The international office and researchers should all be able to access relevant information without requiring technical training. This led to a strong focus on simplicity, and consistency throughout the interface design. Each page follows the same structural logic, visual style, and interaction flow, making sure that when a user becomes familiar with one part of the system, they can easily navigate the rest of it. By doing this we reduce the learning curve needed, allowing a lower bar to participating in the system.

By combining these two main design concepts, scalability and usability, the design is meant to make BIRD a platform that is both powerful enough for complex data manage-

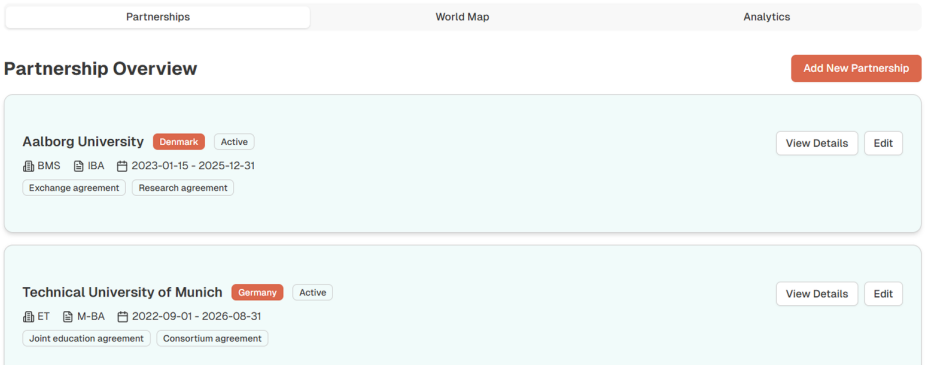


Figure 6.1: Early internal high fidelity design sketch

ment and approachable enough for everyday users across multiple departments within the university.

6.1.2 Design Methodology

For designing the BIRD system, an iterative and stakeholder-centred approach was chosen. This method was chosen because the system was developed in close collaboration with the stakeholders, allowing for quick adoption of changing requirements. By presenting early visual mock-ups and click-through prototypes, feedback was able to be collected throughout the project, ensuring that the design remained aligned with stakeholder expectations.

Early wireframes were created using Figma and were later refined within the development environment itself, making it possible to quickly transform conceptual ideas into click-through prototypes. Based on the feedback received, adjustments were made to the structure, information placement, and overall visual presentation. Updated versions were again demonstrated and reviewed, allowing the design to converge toward the most effective solution. This iterative method ensured that the design decisions were guided not by assumptions, but by validation through stakeholder input. As a result, the final interface of BIRD implements most of the user requirements presented in the requirements section.

In Figma two approaches were used: high fidelity and sectioning. Together immediate feedback both from our stakeholder and our own team high fidelity was used, where the sketch represented the final look, see figure 6.1. Sectioning was later used to layout were components should be placed on a page see 6.4.

6.2 Dashboard Layout and Navigation Structure

The base layout of the BIRD dashboard consists of a sidebar navigation panel on the left and a dynamic content area on the right. The sidebar provides access to the main entities of

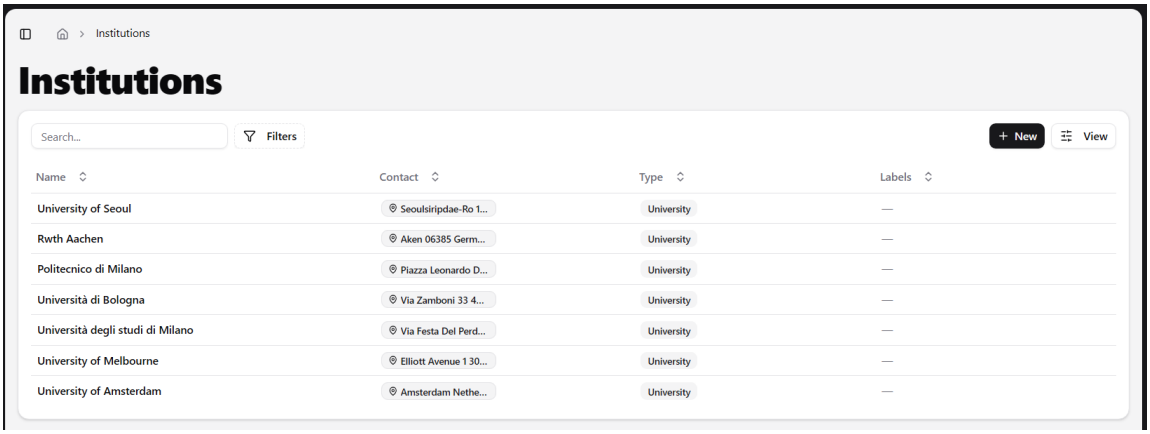


Figure 6.2: Final result

the system, such as persons, institutions, agreements, and networks—while the right-hand section displays the corresponding information and interface components of the selected tab.

The choice for a sidebar layout was made primarily for usability. Given the number of entities and functionalities that BIRD includes, placing all navigation elements in a top bar would have quickly become cluttered and difficult to use. A vertical sidebar, in contrast, offers a clear and persistent navigation structure that remains visible across all pages. This allows users to switch between sections quickly without losing their orientation within the system. From a user experience perspective, this design also minimizes cognitive load by keeping navigation consistent and separated from content, enabling users to focus on the information currently displayed. The final sidebar design can be seen in figure 6.3

The main content area is designed to adapt to the selected entity, displaying data tables, detail views, and interaction components in a consistent layout pattern. This separation of navigation and content supports modularity and allows new entities or features to be added in the future without disrupting the overall structure. The sidebar remains static while the content dynamically changes, creating a predictable and efficient user flow. The main content of the "Institutions" page can be seen in figure 6.2

This base layout, combined with consistent use of spacing, typography, and color themes, ensures that the interface maintains visual coherence across the application. It reflects the project's broader design goals of simplicity, accessibility, and usability, providing users with a familiar and intuitive workspace regardless of their technical experience or frequency of use.

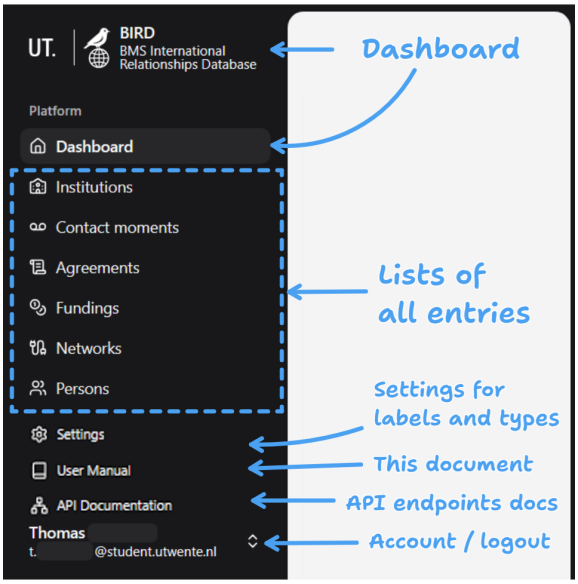


Figure 6.3: Sidebar

6.3 Component Design

For the system to be modular, easy to maintain and customize, a component structure was used.

6.3.1 Component Identification

Component identification was done in layers. The first step was to identify what the stakeholder actually wanted the system to do. As mentioned in chapter 5, this was done by eastblishing the right requirements for the sytem. This was done by a series of meetings. In the first meeting the main entities of the system were identified. These main entities were put in the main navigation bar. In the second meeting we discussed whether this was the right separation of concerns in the various tabs. There was no feedback on this current sidebar, which had the following sections:

- Dashboard
- Institutions
- Contact Moments
- Cooperations (later changed to Agreements)
- Funding

Figure 6.4: Figma component sections

- Networks
- Persons

As illustrated in Figure 6.3, these elements are visible in the sidebar of the interface.

Based on this confirmation we moved down one layer and started to implements what the various sections should be on the various tabs. Herefor we made a Figma layout for the various sections within the tabs for each tab to feed to the stakeholder. In figure 6.4 you can find one of these wireframes.

Here you can see how we display the various sections on the page. In this case it is the Cooperation page where we displayed the various sections on the page. By showing the stakeholder these wire frames we could ask further questions on whether they liked the distribution of sections per tab and if there were still things missing in each section. In the image above, for example, you can see that we have the contact moments above and below the persons, networks, and groups section. The stakeholder agreed to these sections after showing them. Later in this section, we will talk about why they got changed in the end.

After the sections and their positions were determined, we asked the stakeholder what entity information was needed for each section. This was done in collaboration with the backend, in order to extract the right fields for all theentities that belong to those sections. Now per section we could thus start to implement these various fields.

The data for each of the entities was quite important to display. To correctly display all this information, tables are employed for having a nice overview. After identifying the

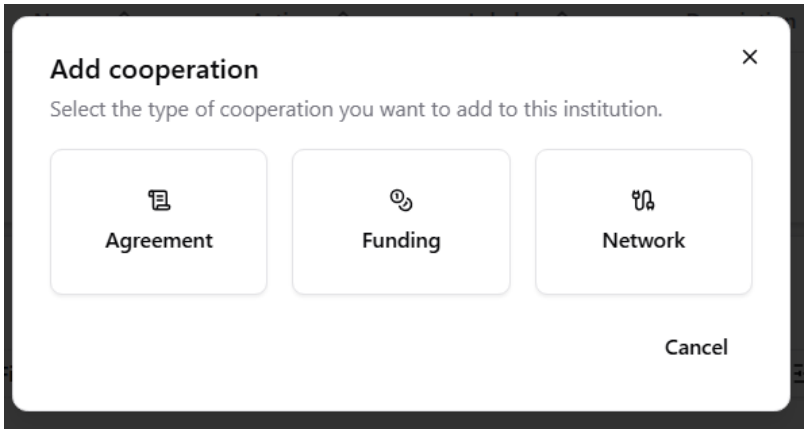


Figure 6.5: Create Cooperation Dialog

various sections with the information they should contain, we moved to implementing the components needed to make the information in each section come to life.

6.3.2 Component Structure

After discussing the different sections with the stakeholders, these had to be implemented using a modular component structure. We followed a bottom-up approach, starting from small base components and combining them to form larger sections. This process looks much like building a pyramid.

For our base layer, we use the SHADCN components as mentioned. This library provides a set of well-designed, customizable components that form a solid foundation for the project. We could easily adapt these components to fit the specific design and functional requirements of our dashboard.

Some key components worth highlighting are:

- Dialog: Used for creating, updating, deleting, and confirming entities within the dashboard. Since these actions occur throughout the application, generalizing them into a single dialog component ensures consistency and maintainability. Any design or behavior changes made to this component are automatically reflected across the entire system. Two dialogs can be seen in figures 6.5 and 6.6
- Data Table / List: The data table is used throughout the dashboard to display entity data. By generalizing it into a reusable table component, we can easily adapt it to different datasets by simply passing the relevant data and fields as props. This significantly reduces duplication and improves consistency across all views.

Each of these larger components is composed of smaller subcomponents, down to the base level. For example: `Dialog Component → Input Sections → Input Forms → Text.`

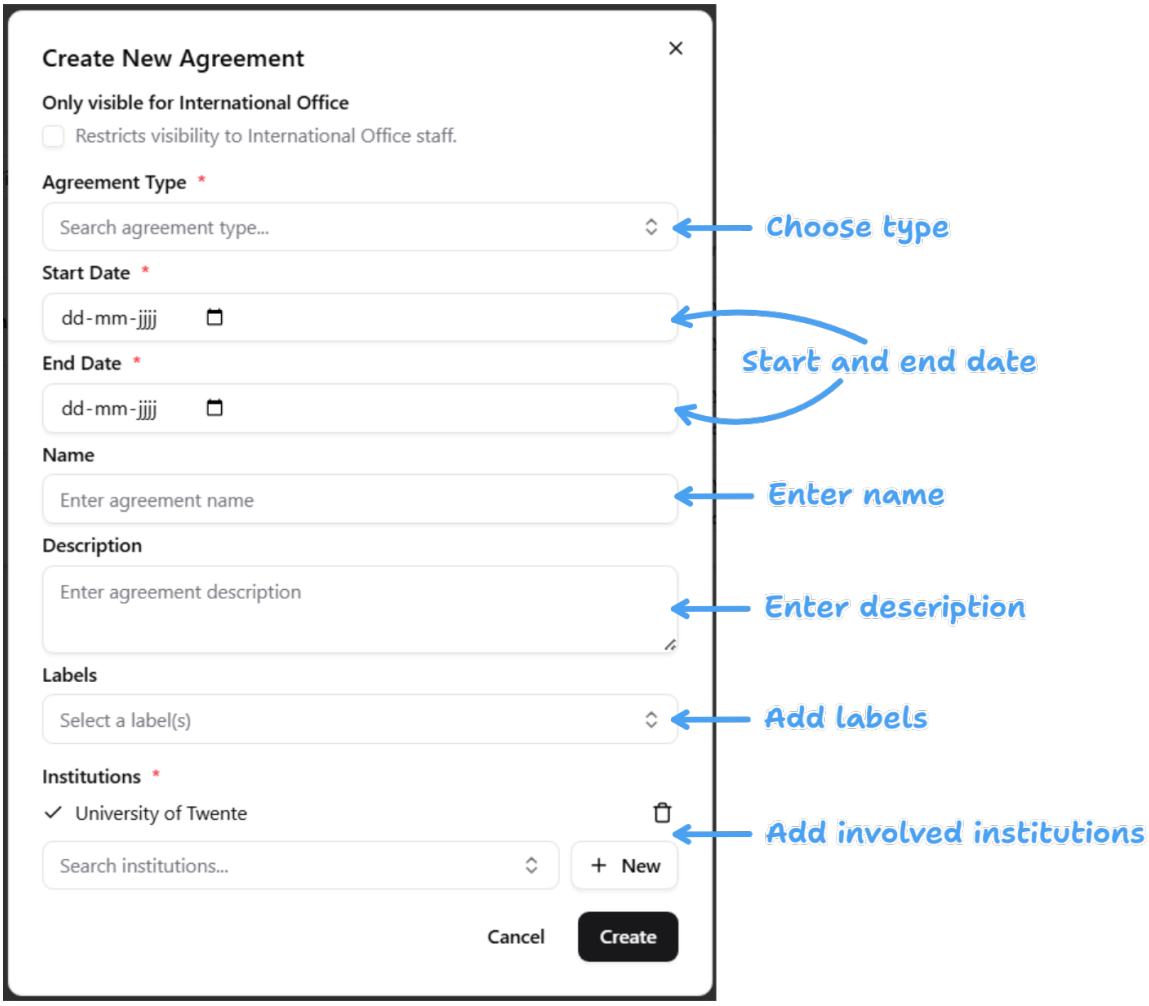


Figure 6.6: Create Agreement Dialog

A single dialog defines the general layout and logic for modal interactions. Inside it, we can dynamically insert various forms that belong to that specific dialog. Those forms, in turn, are built from input fields that handle validation and ensure correct data entry.

This approach provides a flexible and scalable structure that allows for rapid adjustments across the system. If each dialog or table were implemented separately, even a small design or behavior change would require editing multiple files. By using a unified, modular setup, we make the codebase reusable, maintainable, and easily adaptable to stakeholder feedback.

6.4 Feedback and Iterative Refinement

As mentioned before, we held multiple meetings with the stakeholder to gather information about the desired functionality of the system. In each meeting, we aimed to collect feedback and implement it as quickly as possible. The main goal of this process was to create a short feedback loop, allowing the stakeholder to continuously see progress and provide input based on concrete results.

This approach was chosen because stakeholders often have a clearer understanding of what they don't want than of what they do want. By showing them early prototypes, we could refine the system step by step based on their reactions and insights.

6.4.1 Initial Stakeholder Evaluation

In the first meetings, we focused on identifying the core entities and functionalities of the system. Based on this information, we created a set of wireframes that served as the first visual reference for the stakeholder. These wireframes allowed us to discuss layout, structure, and initial functionality before diving into full implementation.

After building the first interactive sections and integrating them into the different tabs of the application, we held another feedback session. During this session, several new requirements and changes emerged as the stakeholder gained a clearer understanding of the system's behavior and possibilities.

6.4.2 Design Revisions

During the development phase, it became clear that some requirements were more complex than initially expected. For example, the stakeholder decided that the "Contact Moments" on the "Agreements" page were unnecessary. Similarly, the "Groups" section was no longer needed as a separate entity; instead, people could simply be labeled as belonging to certain groups. This labeling made it possible to see group memberships when viewing a person's profile, without requiring an additional section.

After each meeting, we demonstrated the latest version of the system to the stakeholder, who could then explore the interface themselves. This hands-on testing helped them better

understand how different flows worked and whether these matched their real-life processes. These user-testing sessions were especially valuable, as they often led to new insights and refinements that improved both usability and alignment with their workflow.

Importantly, these revisions did not have a major impact on the frontend architecture. Most adjustments only required changes to the props passed into existing components rather than rewriting the components themselves. This modular structure allowed us to adapt quickly to stakeholder requests, ensuring efficient iteration and smooth progress throughout the project.

6.5 Final Interface Overview

6.5.1 Current Design

The final design of the system brings together all core functionalities into a clean and structured interface. The dashboard consists of multiple main sections, each representing one of the core entities such as Institutions, Persons, Agreements, and Contact Moments. These are accessible through a navigation bar that allows users to switch between sections easily.

Each section presents data in a data table, which provides a clear overview of all entries. Users can create, edit, or remove entities directly from the same view using modal dialogs, avoiding unnecessary page reloads. This layout ensures that the most important actions are always visible and within reach.

The dialog components play a central role in the design. They are used throughout the system for actions like creating or editing items, and they follow a consistent layout and structure. By keeping the interface modular, every section of the dashboard maintains the same look and feel, reducing user confusion and simplifying future updates.

In addition, the interface is responsive, meaning it adjusts to different screen sizes without losing clarity or usability. Even though the system is primarily intended for desktop use, basic responsiveness ensures it remains functional on smaller screens as well.

Overall, the final design focuses on simplicity, efficiency, consistency and scalability, offering an intuitive experience that matches the administrative workflow of the stakeholder. The final design can be seen in figure 6.7

6.5.2 Usability and Visual Coherence

The interface was designed with usability as a top priority. We aimed to make all interactions intuitive and self-explanatory, minimizing the learning curve for new users. The use of consistent colors, typography, and spacing ensures a coherent look and feel throughout the dashboard.

This was mostly achieved by asking the stakeholder for the definitive terms used by their colleagues when using the system. In combination with applying naming conventions

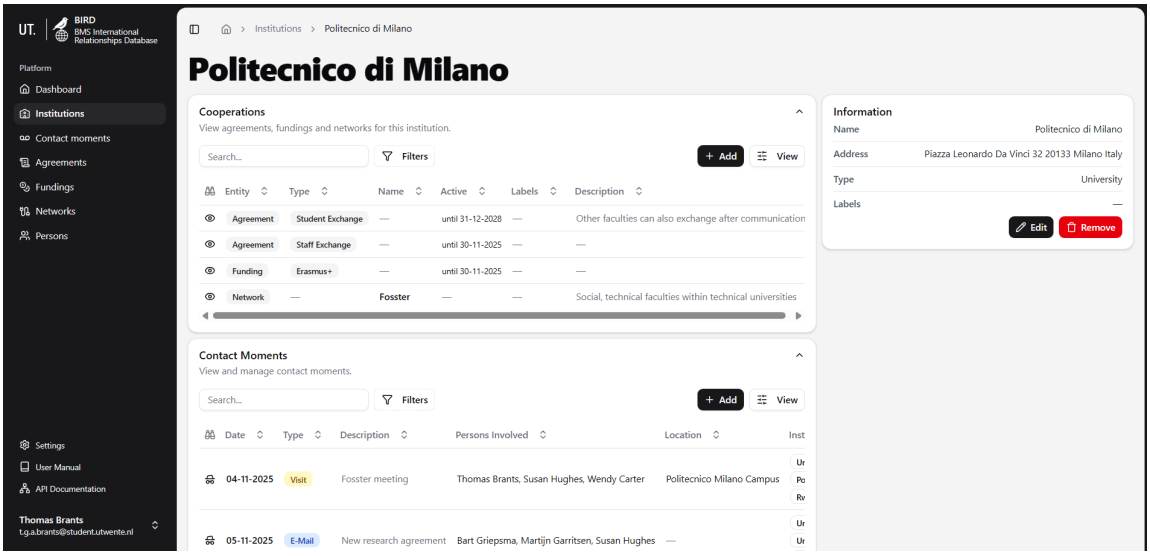


Figure 6.7: Final Design

which imply their meaning such as "new" ,"add", and "edit", we allow for a trivial workflow as much as possible.

The visual style follows a minimal and modern approach, using the nautral colors of black and white along with subtle gray highlights to keep the focus on the data rather than on decorative elements. Interactive components, such as buttons, input fields, and dialogs, follow a consistent pattern based on the SHADCN UI library, with primrary and secondary colors. Allowing for visual unity and usability.

During stakeholder testing sessions, users reported that the layout felt logical and easy to navigate. The separation of entities into different tabs helped them quickly locate the information they needed, while features like search and filtering made working with larger datasets more efficient.

Chapter 7

Development Process

7.1 Backend Design Rationale and Approach

7.1.1 Overall Design Goals

The backend of the BIRD system was designed to provide a robust, secure, and maintainable foundation for managing relationships between institutions while maintaining data integrity and traceability.

Scalability ensures that we can support the growth of new data that is added over time, but also the big influx of data that will be imported at the beginning of use. By structuring the system around modular Django applications, each component can be worked on independently without affecting the others apps in a major way. This modular approach is good for both database and API scalability, enabling future extensions such as analytics.

We achieved modularity by using django to separate classes across models, serializers, and viewsets. Each model like 'institution', 'cooperation' or 'person' contains its own data and logic. This structure not only improves the readability and maintainability of the code but also simplifies collaboration among developers. These separated models allow us to create a standardized REST API.

Security was a critical focus because the system manages institutional information from the University of Twente. Authentication and authorization mechanisms ensure that only authorized users can modify or view specific data. Django also has built-in protections against SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF), this already makes the backend quite secure without us even adding extra security features.

7.1.2 Design Methodology

The backend was developed in the same way as the design process, using different iterations to build up. This approach allowed the team to implement and refine the backend

components based on stakeholder feedback.

Regular meetings ensured that the backend structure is aligned with the stakeholder's wants and the long-term goals of the system. This design methodology made the backend adaptable to changing requirements while maintaining a good standard of code quality.

7.1.3 Choice of Frameworks

We built the backend using Django, combined with the Django REST Framework (DRF) for our API development. Django was chosen after a meeting with our supervisor, since he would like consistency of the multiple projects he maintains. Its Object-Relational Mapping simplifies database management and allows for easy to use complex relational modeling.

The Django REST Framework extends Django's capabilities by giving us access to serialization, authentication, and automatic API generation tools. This enables quick development of RESTful endpoints that integrate well with the Nuxt frontend.

For authentication, BIRD has the University of Twente's Single Sign-On (SSO) system for the login system. This setup lets staff and researchers log in using their existing UT accounts, ensuring secure and centralized authentication management. It also helped us implement clear role-based access control by using the user their faculty mentioned in the display name.

For our database, SQLite was selected for its lightweight, file-based architecture and zero-configuration setup. This was also done after a meeting with our supervisor, also to ensure consistency across projects. It supports all required relational features and can be seamlessly replaced by PostgreSQL or MySQL in future scaling scenarios without requiring significant code changes.

Together, Django, OAuth2, and SQLite create a secure, modular, and maintainable backend that aligns with the design goals of BIRD.

7.2 Initial database Design

7.2.1 Table and field Identification

For the initial design of our database, we had not yet determined with our main stakeholder what fields would be required to create the application they had wished for. This is mainly the reason most tables in our initial database design in figure 7.1 look rather empty. All fields currently come from our own understanding of the product based on earlier experiences and logical thinking. These fields will later be expanded upon.

The table structure had also not entirely been perfected at this time. All tables shown below in figure 7.1 we had gathered from our initial project proposal without having discussed our interpretations with our stakeholder yet.

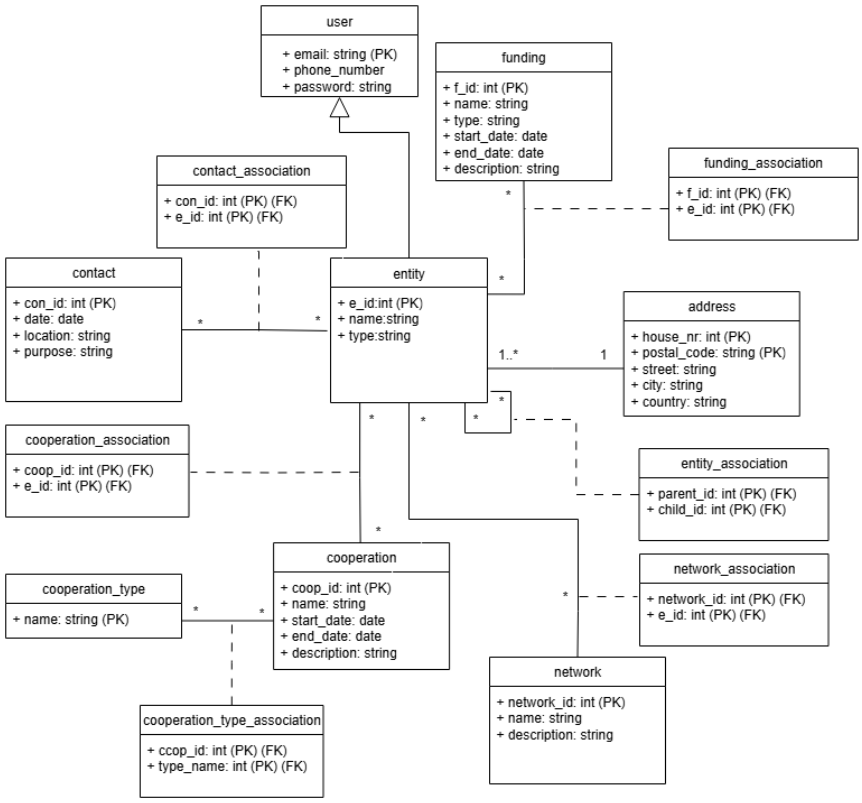


Figure 7.1: First database version

7.2.2 Initial database Structure

As shown in figure 7.1 our initial design in centered around the entity table. This table has the person, faculty and institution tables all in one. It also allowed for multiple different entities to have the same address. This was not needed in later instances since the only entity that needed an address was the institution itself and there were no different institutions making use of the exact same address. With this general entity approach we also had the need for a many-to-many association for the entity table to itself since a person (entity) could be part of a faculty (entity), which could be part of a institution (entity).

You can also clearly identify multiple different association classes in figure 7.1. Even though these are automatically handled within Django, in our first version we decided to show them for good oversight during the start of the project, but in later versions they will not be present in the schema.

7.3 Feedback and Iterative Refinement

7.3.1 Design Revisions

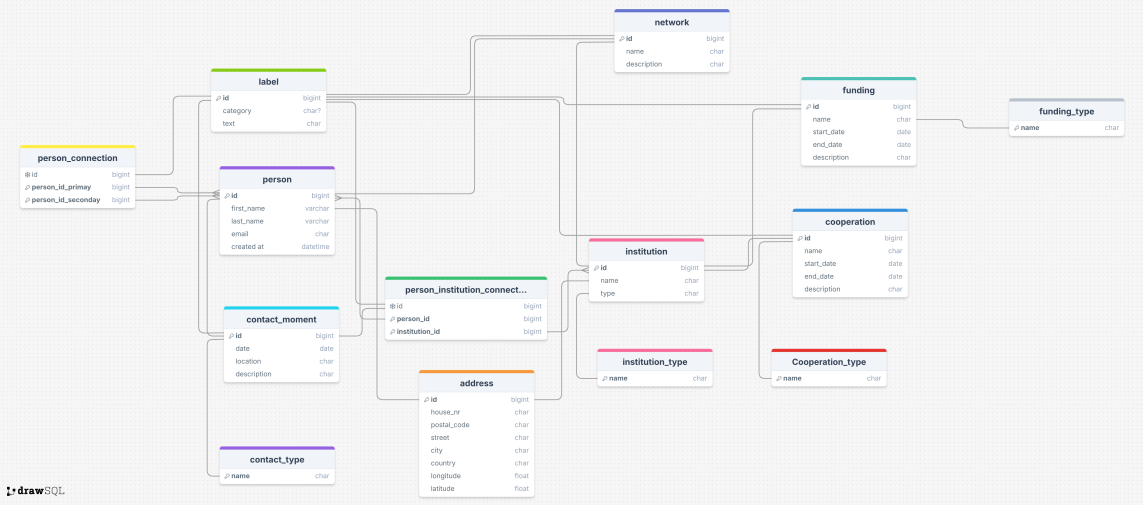


Figure 7.2: Second database version

In figure 7.2 we have made quite a number of changes to our initial design after evaluating it with our stakeholder and supervisor. After some consideration we had come to the conclusion that a person is not as important in our system as an institution. With faculties being even less important. With this feedback we have decides to split up our entity table into a person- and an institution table, with an association table in between since a person can be part of multiple institutions. The deletion of faculties have sparked us to create a label system instead which is linked to almost all tables in the database where you can specify not only under which faculty a certain action or person falls, but also what study level it is for or other functions yet to be worked out by the BMS international office.

Furthermore we have also added more type tables for the 'contact moment', 'funding', and 'institution' as they were fields in their respective tables at first. This was changed to make it easier to manage all the different types.

After one of our meetings we also gathered an additional requirement, namely that the main stakeholder would also like to see an overview of which people an individual is connected to, hence we added the 'person connection' table to make this a possibility.

The last adjustment we made was the addition of longitude and latitude fields to the address table to enable us to show all institutions on a map on our dashboard.

7.4 Final Database Overview

7.4.1 Current implementation

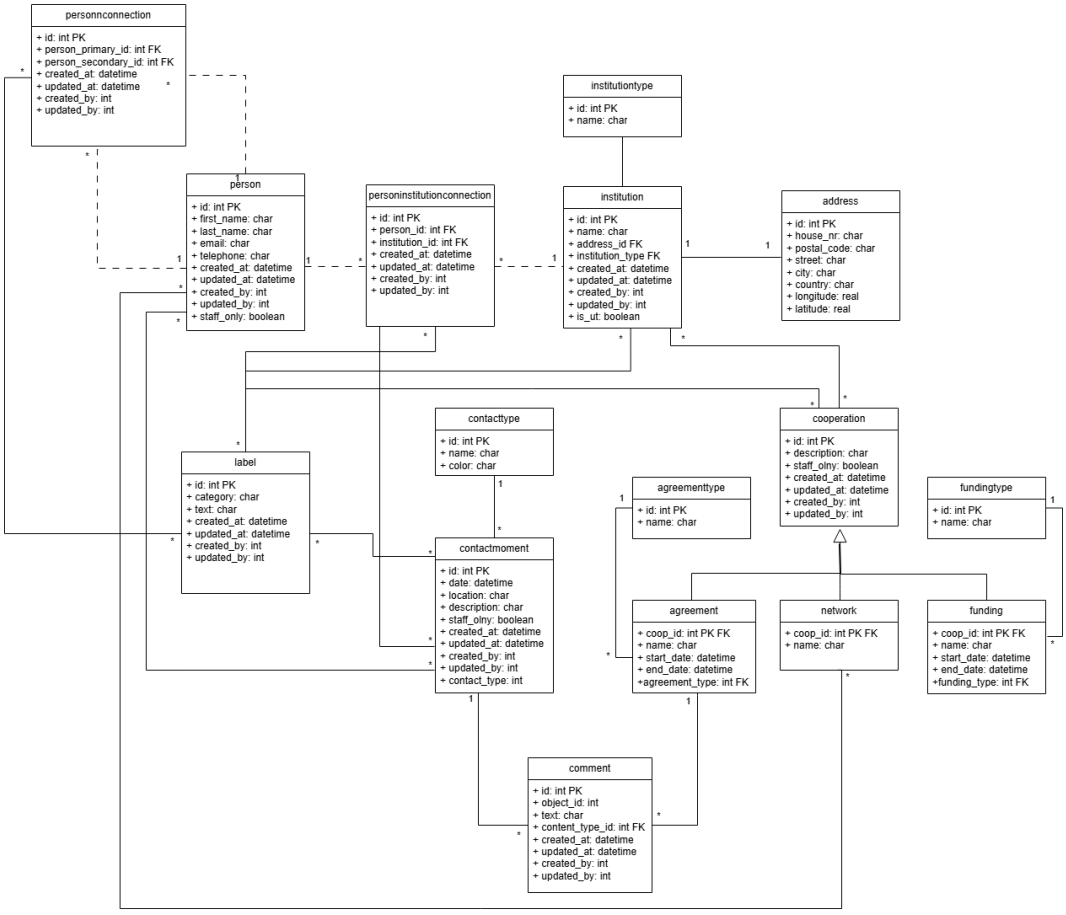


Figure 7.3: Final database version

Above in figure 7.3 you can see our final database design that has been iterated upon a handful of times. The first difference is the grouping of networks, funding, and agreements (formerly known as cooperations) into subclasses of one 'cooperation' superclass. This was done to make working with these classes easier, since they already had quite a few attributes in common.

Additionally, we have finally worked out all the necessary fields for all of our tables.

Most importantly you can see the 'created at' 'updated at' 'created by' 'updated by' fields in multiple different tables, these are to enforce data traceability and accountability. We also added the 'is UT' boolean to our institution table, since the University of Twente is the institution our whole system is centered around and thus has special behavior. Lastly, our 'contactmoment' and 'cooperation' tables have been given a 'staff only' boolean to ensure proper privacy for users if they wish.

Lastly, we added an additional table called 'comment', with which you can leave a comment on a contact moment or an agreement if you wish to add additional information or clarify some statement.

7.4.2 API endpoints

The system uses a RESTful API for communication between the frontend and backend. All endpoints exchange data in JSON format and are protected by authentication using JSON Web Tokens (JWT).

Through these endpoints, users and other systems can manage institutional data, agreements, contact moments, and related resources. For these endpoints we used a library called 'djangorestframework' to serialize the database models to JSON. Authentication is primarily handled via Microsoft OAuth (Single Sign-On) for users. For this we used a library called 'allauth'.

For an extensive list look at Appendix C.

7.5 Deployment

The final step in the development process was to deploy the application. To accomplish this, the team was granted access to a dedicated namespace within the University's Kubernetes cluster. A GitLab runner was created and linked to the project's repository so that whenever a new version of the code was pushed, a new container image would be automatically built.

During the cluster setup, several challenges arose. Due to stricter security policies implemented by the University of Twente following cybersecurity incidents at other institutions, containers were no longer allowed to read or write data internally. As a result, we had to find an alternative method for data storage. Our solution was to mount an external volume to the container, providing a secure location for data without storing it inside the container itself. An additional benefit of this approach was data persistence: we could redeploy the application without losing any stored information.

7.6 Frontend Architecture and Implementation

Earlier, we discussed how we actually established final design of the system. In this section we will discuss what tools and architecture we used to actually bring the design of the

frontend to live.

7.6.1 Choice of Framework

One of the requirements from the maintainer was to use Vue. Non of our team had prior experience with Vue. There was experience with React and specifically NextJS. The framework Nuxt is inspired by NextJS, thus the decision was made to use it.

Nuxt provides native support for server-side rendering and static site generation, which contributes to faster load times and improved usability when accessed through a web browser. Since BIRD is designed to be a browser-based web application, this framework enables users to open and operate the system without the need for local installations or special configurations, aligning with the goal of making the platform accessible to all university staff.

For development, Nuxt uses Vite, a lightweight build tool that provides hot-reloading and rapid compilation. This greatly accelerates the design iteration process, as visual or structural changes in the code can be viewed immediately without rebuilding the entire project. In addition, the integration of TypeScript offers static type checking and compile-time validation, which reduces runtime errors and improves overall code reliability.

The frontend leverages Zod, which is a TypeScript-first schema validation library, to ensure data integrity both at runtime and compile-time. Zod schemas are defined for all entities in the application, providing validation rules that are enforced through Vee-Validate when users interact with forms. This dual-layer validation approach, where Zod schemas validate data before submission and the Django REST Framework backend performs additional server-side validation, ensures that invalid data is caught early in the user interaction flow, providing immediate feedback and reducing unnecessary API calls. When input is not provided incorrectly, Zod provides the correcting functionality to notify the user of the incorrect input. Furthermore, Zod's type inference capabilities automatically generate TypeScript types from these schemas, creating a single source of truth for both validation rules and type definitions. This integration means that validation errors are caught not only at runtime through Zod's validation, but also at compile-time through TypeScript's type system, as the generated types ensure that components and API functions receive data in the expected format.

Together, these technologies create a modern, maintainable, and efficient frontend environment that supports the iterative design process and ensures that the BIRD interface remains scalable, consistent, and user-friendly across different usage scenarios.

7.6.2 Architecture

The goal of the frontend was to be composable, reusable and maintainable. This meant writing multiple small components to be reused by multiple bigger components. In this section the most important and complex components will be explained further.

Dialogs

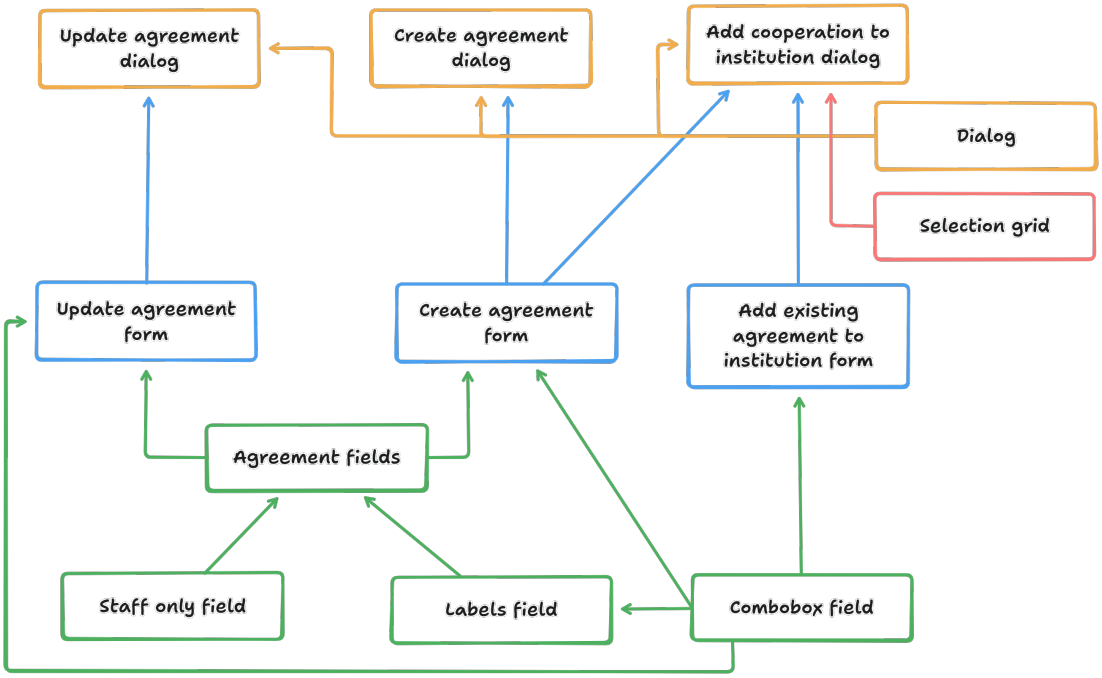


Figure 7.4: Dialog components for agreements

A good example is the dialog system as seen in figure 7.4. The smallest components are the fields in green. Fields can be composed together to create bigger field sets, such as "Agreement fields" or forms. Fields them self only contain logic on how to input and visualize data. The next layer of components is are the forms, in blue. Forms use fields to show the inputs and contain logic on actually mutating the data via the REST API. The forms also validate the data on the client side before sending a request to the API using ZOD. This allows for more specific error message, and possible in the future in multiple languages. The layer above are the dialogs them selves, in orange. These only contain logic for opening and closing. Or sometimes as in the "Add cooperation to institution dialog" logic for choosing which form to show.

Data Table / List

The next big component is the List component as seen in figure 7.5. The list component will be imported on a page. The page will supply all the data / rows with a type given, such as institution or person. The list component will pass this data to the green sections:

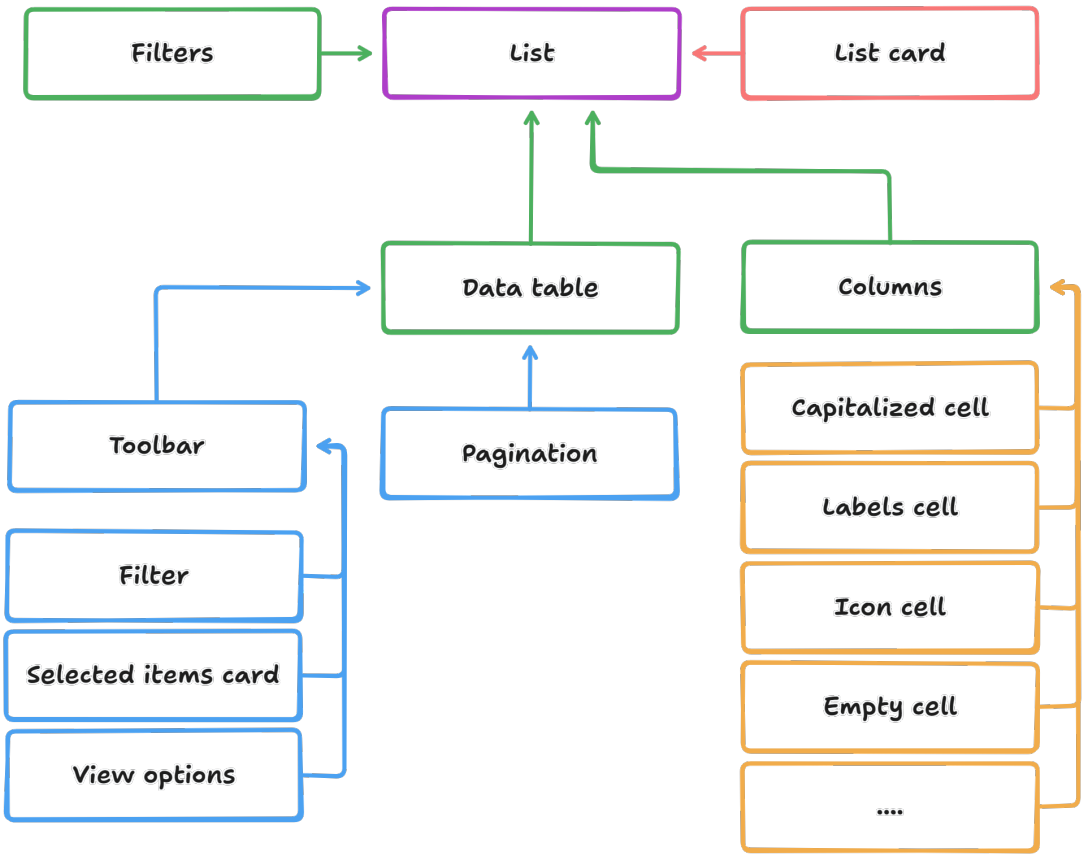


Figure 7.5: Data Table / List components

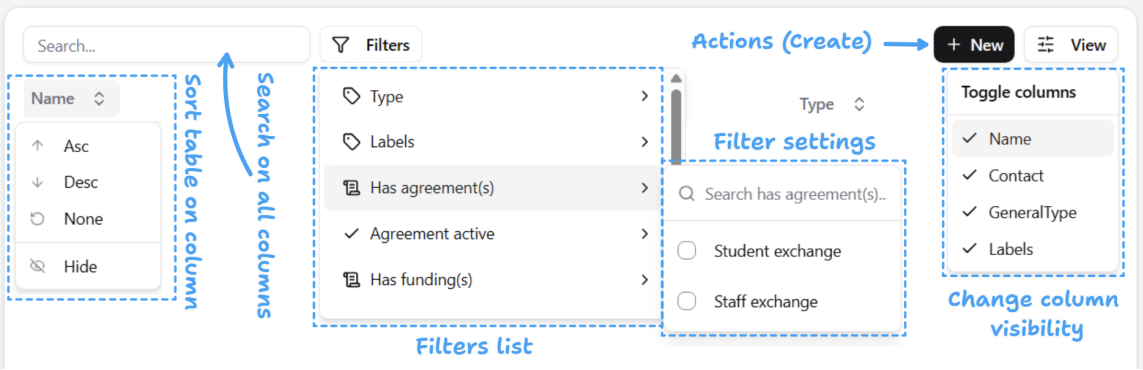


Figure 7.6: Table final design

columns, data table and filters. Based on the data given the column provider will get the correct columns. Each column is split up into a definition and a cell. The definition handles the data and passes it to the cell for visual representation. Then these column definitions will be passed to the data table to show the table. The data table is visual component that renders the table using TanStack Table. The data table is split up into toolbar, the table it self, and pagination. The toolbar will be shown at the top has a search bar, filter button and view options. From the list component it is also possible to pass custom action buttons which will be shown next to the view options. The pagination is shown below the table, it allows the user to navigate table pages and choose the size of those pages. The filters section will retrieve all filters that can be applied to the chosen columns. This way the filters are dynamically shown so the consumer of the list component does not have to supply them. Figure 7.6 shows the final look of the list component.

API communication

To communicate with the REST API TanStack Query was chosen to handle both retrieval and mutation of data. TanStack Query "makes fetching, caching, synchronizing and updating server state in your web applications a breeze". When data is retrieved using a query the data is cached and automatically re-fetched when mutating data or when a page has become active again after time of inactivity. Our usage of it can bee seen in figure 7.7. To perform each database request $fetch from NuxtJS is used under the hood. Around this fetch a composable was written to handle authentication, in blue. Actions, in green, use this composable to make requests. Each action is than wrapped in a useMutation or useQuery depending on what type of action it is. The mutation also call 'invalidateQueries' to invalidate query caches and start fetching.

```
const { data: agreement } = useAgreement(agreementId)
const {
```

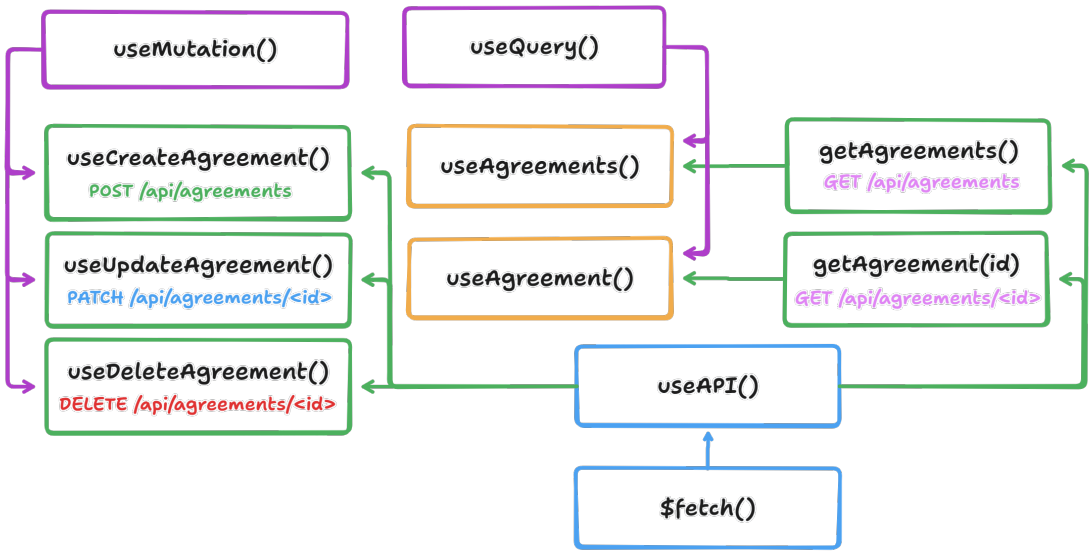


Figure 7.7: API Actions

```
    mutateAsync: removeInstitutionsFromAgreement,
    isPending: isRemovingInstitutions
} = useRemoveInstitutionsFromAgreement()
```

7.6.3 Frontend Approach

Three people within our group were put on the frontend of the application. Using JIRA we were able to divide the tasks into the separate components and sections identified. This enables us to work on different things and combining them eventually into the final sections. To increase development speed the components were divided based on difficulty and experience of the developer.

Chapter 8

Testing & Evaluation

8.1 Validation & Sanitization

Django provides a framework for validating input through serializer classes. These classes offer methods for both field-level and object-level validation. In this project, validation has been used across all serializers to ensure the integrity and security of data.

To protect against security vulnerabilities such as XSS attacks, a Python library called Bleach has been integrated for sanitizing text input. A custom Bleach serializer mixin has been created. This mixin has been implemented in all serializers that have some kind of text fields. This mixin removes HTML tags and embedded scripts from the text inputs, making sure that any user input is sanitized before being stored in the database.

Other than text sanitization, validations are enforced for data correctness. Required fields are checked if they are empty to ensure that they exist. This guarantees that the required information is captured and prevents incomplete records from being created. Text fields, such as names and descriptions, are checked for length constraints. They need a minimum character count to avoid trivial or inadequate input.

Relational consistency and uniqueness are also checked. Relationships, such as those between people and institutions, are validated to prevent duplicate entries, circular references, or self-referentials between persons. Lists that are intended to contain multiple entries, such as an agreement's institutions, are checked to make sure they contain at least one element, preventing the creation of records with insufficient relational data.

Date fields are also validated for consistency. Start and end dates are compared to ensure that they are chronological, preventing cases where an end date is before a start date.

Additionally, access control rules have been created to hide sensitive information when a user is not allowed to see it. Objects marked with a staff-only tag that contains sensitive data are hidden or anonymized in the serialized output, depending on the permissions of

the requesting user. A staff person can see all data, while a regular BMS user can only see what is not marked as staff-only, or what they themself have created.

8.2 Tests

This project used multiple testing strategies. One of them was the use of Django's built-in TestCase class, and another was manual checks using Postman or interacting with the frontend. The TestCase tests were used to verify the validity of serializers and models, ensuring that appropriate response codes and messages were returned. Almost all serializers were tested for the correctness of their fields, such as that required fields could not be empty, that text fields met minimum length constraints, and that relational fields maintained uniqueness and consistency. Duplicate entries, self-references, and relationships were tested to ensure data integrity.

Sanitization of input fields was verified through tests that prevented potentially harmful HTML or script content, such as <script> tags, into text fields. The serializers' Bleach mixin was checked to confirm that these tags were removed while retaining the valid content.

Date-related validations were tested to ensure logical consistency, such as confirming that end dates did not precede start dates in agreements or funding records. Lists of associated objects were checked to ensure they were not empty when at least one element was required.

Signal behaviors were tested by simulating external events, such as social login actions. These tests verified that new users were correctly linked to existing person records or that new person instances were created when necessary. Changes in external user data were also checked to ensure they correctly updated the associated person objects.

As mentioned before, there are test cases to verify the field-level validation, examples are name and description length validations. Additional test cases ensure that empty or null values are not allowed when required.

Some manual tests focused on user permissions. Three different user types were tested: a BMS user, a staff user, and an unauthenticated user. The unauthenticated user was verified to be unable to perform any actions. The BMS user was tested to ensure access was limited to their own objects and other objects they were allowed to access. Finally, the staff user was tested to confirm they had full control over all objects and actions.

Chapter 9

Final Product

9.1 Final BIRD System Overview

Our final product named the BMS International Relationship Database (BIRD) is a user-friendly platform where you have a good overview of the international cooperations of the UT.

Through BIRD, users can view cooperations and contact moments in an easy to understand way. Each cooperation includes details such as the type or involved institutions and people. Users can also view comments and contact moments, such as meetings, visits or emails related to a collaboration.

The dashboard offers an overview of all partnerships. It includes numerous filtering options that allow users to quickly sort information based on certain criteria, such as name, labels, or type. This enables users to locate specific partnerships or find cooperation patterns very easily.

Authorized users, such as members of the International Office, have the ability to add, update, and remove database entries and labels directly within the application. This ensures that partnership information remains accurate and up to date without requiring any technical background. New information can be registered through the application, where the user is allowed to specify institutions, involved persons, and cooperation types.

Additionally, there is a commenting feature which enables users to add notes with context or updates to specific cooperations or contact moments. This improves internal communication, because notes, comments and follow-ups can be stored alongside the relevant data instead of being stored some other place.

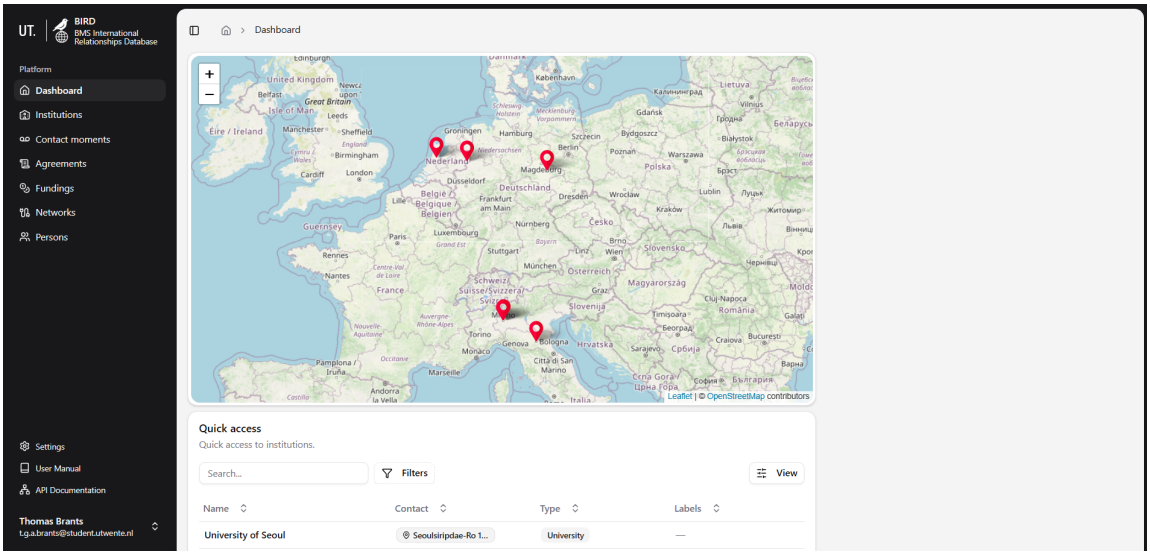


Figure 9.1: Final dashboard

Finally, the BIRD dashboard offers a world map with all (inter)national partnerships, giving users a nice overview of the UT's global cooperations. This overview, together with the search and filtering features, allows users to quickly navigate through the university their international network.

By focusing on the usability of the application, the final BIRD system provides a application for managing and exploring the university their international relationships, helping them strengthen, categorize and expand its global partnerships with ease.

Chapter 10

Discussion

10.1 Areas of improvement

Even though the project went very smoothly over the last ten weeks, there are still parts of the process that could be improved upon in later projects.

The first point we want to discuss is the communication with our supervisor and stakeholder, even though we communicated in our group really well, this did not always translate further. Especially in the first phase of the project when we had not set up a proper meeting structure, there were some issues regarding informing all the required people, but later on in the project we were more aware of this problem and tried to limit it as much as possible. In future works of ours we would assign the communication with outside parties to one or two individuals to erase as many miscommunications as possible.

Furthermore, if we were to do the project again, we would have started using the scrum method earlier in the project. We had already started using an application for this (Jira) at the kick off of the project, but did not regularly start having scrum meetings until some weeks later. This meant that the beginning was a bit hectic and it was hard to see who was working on what. After we implemented the scrum system more strictly it drastically increased our productivity and oversight of the work that was yet to be done.

Lastly, we want to mention our ways of requirement gathering. Our main stakeholders knew roughly what they wanted, but they were not sure of the details yet. So we should have spent more time in the beginning analyzing the different flows through the system so they would have an easier time expressing their wants instead of our approach where we built a system first and then asked our stakeholders in a meeting what they would like to be different. That way we could have saved valuable time to use on improving BIRD even further.

10.2 Future Improvements

As the maintenance of the BIRD system will be inherited by our supervisor after the conclusion of the module the future improvements are not in our hands, but there are parts that we have already work on, but we did not manage to finish in time. Namely, the timeline functionality. A functionality where you can easily see the updates that have bee made to entities like institutions, which could then be shown on their respective pages. The API endpoints and data structures on the backend are already in place for this, but we lacked time to fully work this feature out entirely, so this could be a possibility for future work.

- **Notifications**: A future version of the system could include a notification feature to keep users informed about updates or changes. This would help increase user interaction with the platform by actively pulling them back into the system. The more users engage with the platform, the more valuable and dynamic it becomes for the entire community.
- **Activity Timeline**: During the initial stakeholder meetings, the idea of a timeline feature was discussed. Such a timeline could display key activities within the system, for example when an institution joins or when a new agreement is created. This would make the platform feel more alive and transparent, showing users that the system is actively being used. A visible sense of activity can also help maintain user engagement over time.
- **Reducing Repetitive Code**: Although the current implementation already makes extensive use of reusable components, there is still room to reduce repetitive code further. Future refactoring could focus on modularizing common logic and improving code abstraction. This would make the system easier to maintain, more efficient, and cleaner in structure.
- **Multi-Language Support**: Currently, the entire system is written in English. Adding multi-language support would make the platform more inclusive and adaptable for international use. This could also make it appealing for other universities or organizations that operate in different languages.
- **Framework Considerations**: While Nuxt served its purpose during development, it did not fully meet all expectations. Nuxt dev server had a slow cold start, hot-reloading did not always work for each component. Sometimes it would also do a full reload, even when just typing information into a form. For future iterations, it could be beneficial to return to plain Vue.

Appendix

A

Manual

BMS International Relationship Database

User Manual

UT.

UT.

1. General Information

BIRD is a web application to provide an centralized system that allows research and educational staff and the International Office (IO) of BMS to access and manage information about collaborations with partner universities worldwide.

BIRD can be accessed on https://bird.apps.utwente.nl/

2. Authentication

BIRD uses the Single Sign-On service of the University of Twente. To authenticate you must be in possession of a valid **BMS** account of the University of Twente. Exceptions to this rule can be made by the system admin.

To login go to https://bird.apps.utwente.nl/, it should show a page similar to Figure 1. Click “Sign in with UT”, you will then be directed to the Single Sign-On of the UT. After successful authentication you will land on the “Dashboard” page.

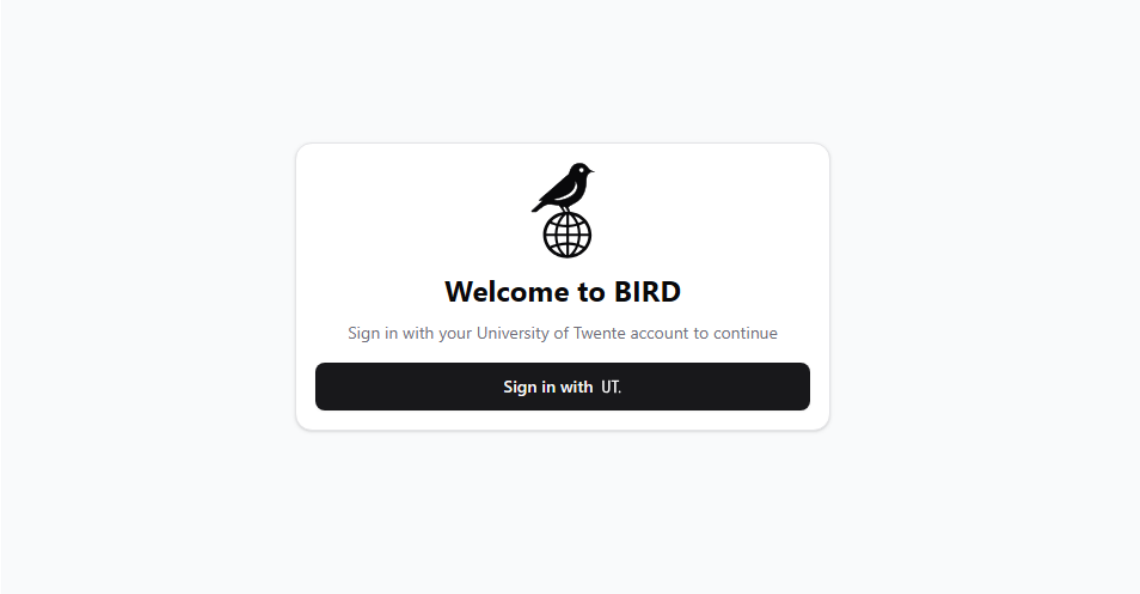


Figure 1

3. Main concepts

BIRD exist out of 5 main concepts. These concepts together give insights into communication and cooperations with other organizations.

3.1 Institutions

Organizations we collaborate with (universities, municipalities).

3.2 Persons

Persons that are either part of BMS or contact persons from other institutions. Users from BMS will also be linked to a person.

3.3 Cooperations

Cooperation is a general term for agreements, networks and funding.

Agreements: An "contract" between two or more institutions for example for "student exchange"

Networks: An collaboration between two or more institutions. Most of the time with a specific goal in mind.

Funding: An object to keep track of where funding comes from.

3.4 Contact moments

Logged interactions such as emails, meetings, or calls, including date, persons, comments.

3.5 Labels

Can be used to categorize, all types of objects in the system. Examples are "Study Levels" for agreements. Or "Faculty" for persons.

4. User Interface

4.1 Sidebar

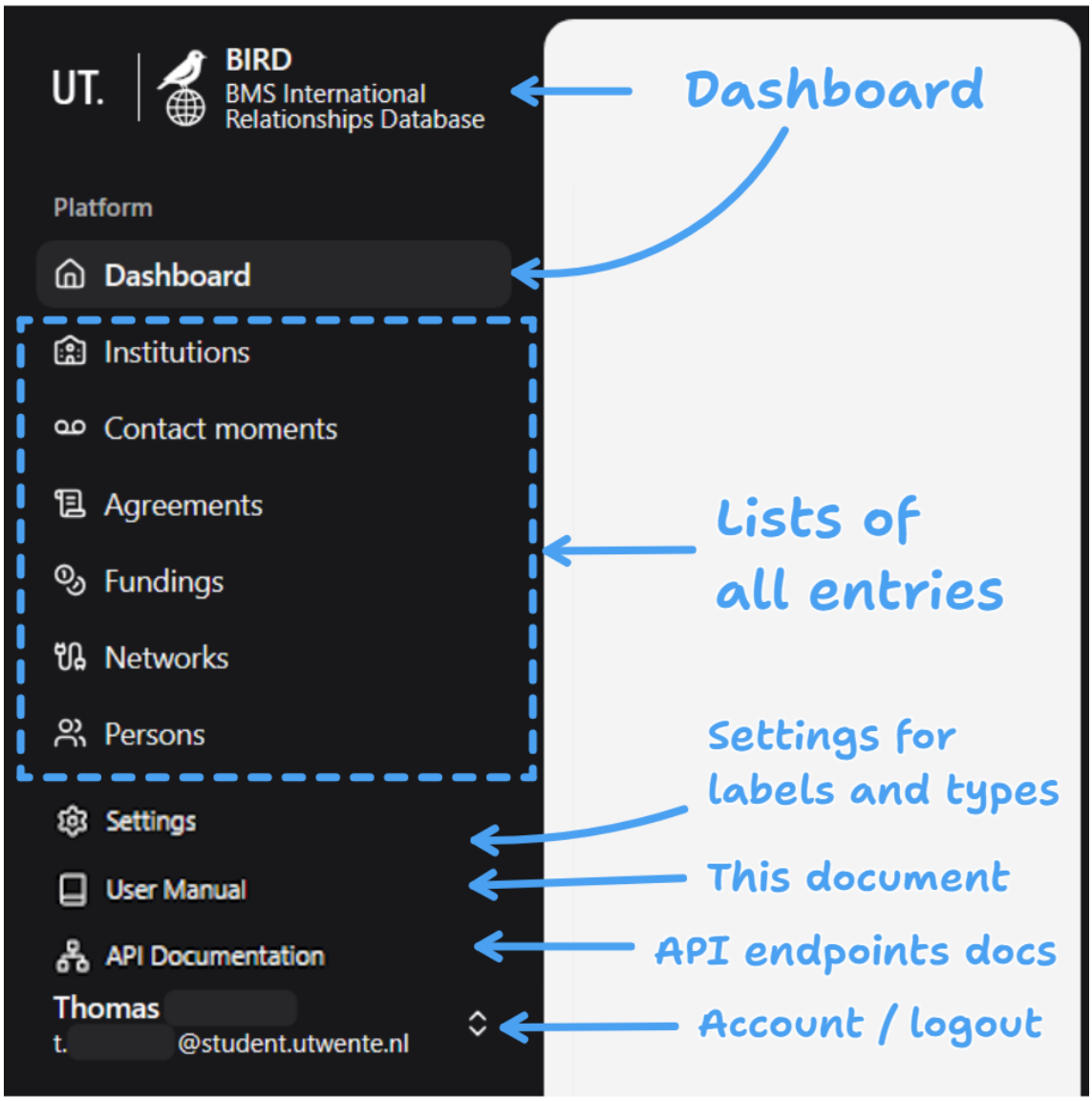


Figure 2

4.2 Topbar

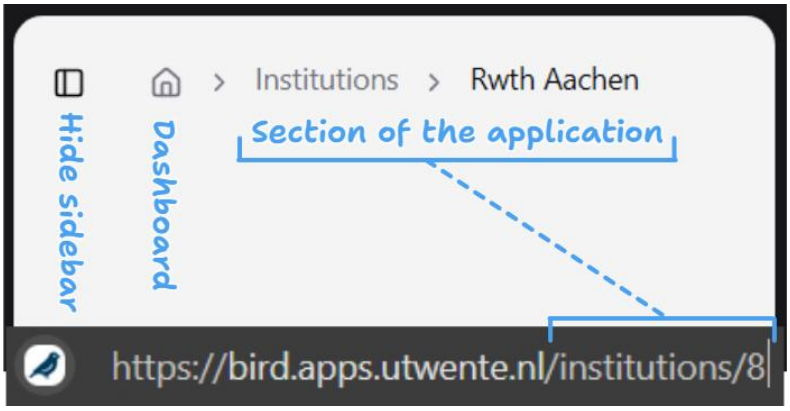


Figure 3

4.3 Tables

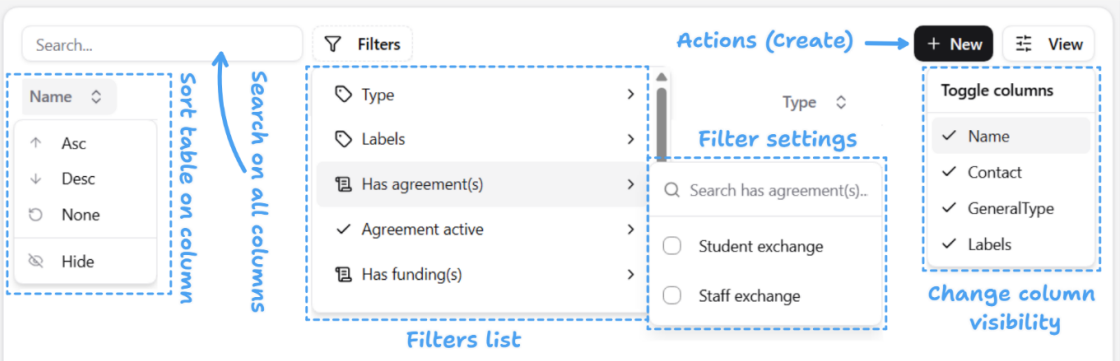


Figure 4

Selection

Some tables have the possibility to select one or multiple rows. When a selection is made a toolbar will pop-up. From this toolbar actions can be performed.

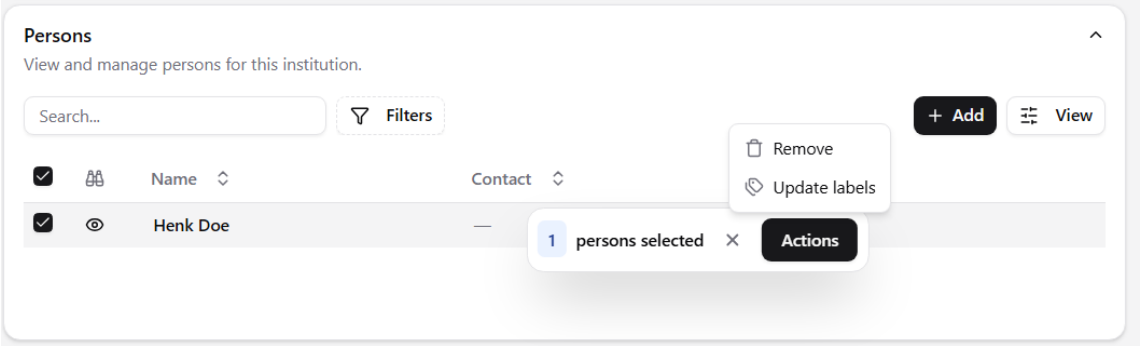


Figure 5

Figure 6

4.4 Information Cards

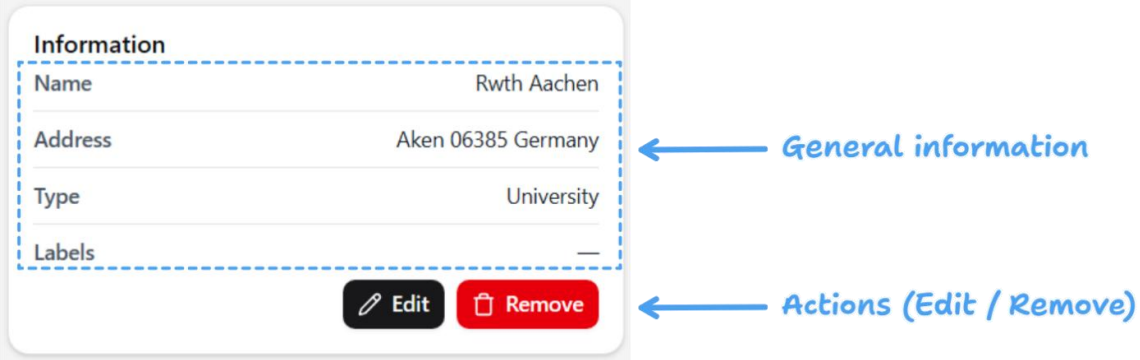


Figure 7

5. Permissions

In the BIRD application only two groups exist: staff and non-staff.

Staff
Can add, edit, remove and view all data in the system.

Non-staff
Can add any data to the system. However they can only edit and remove data create by them selves.

Non-staff have the option to make data only available to them selves and the staff users.

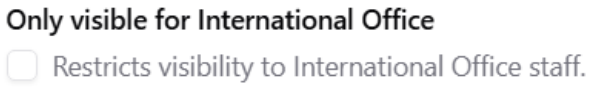


Figure 8

In tables where rows are visible for you but not others this icon will appear in front.

Hidden

Visible

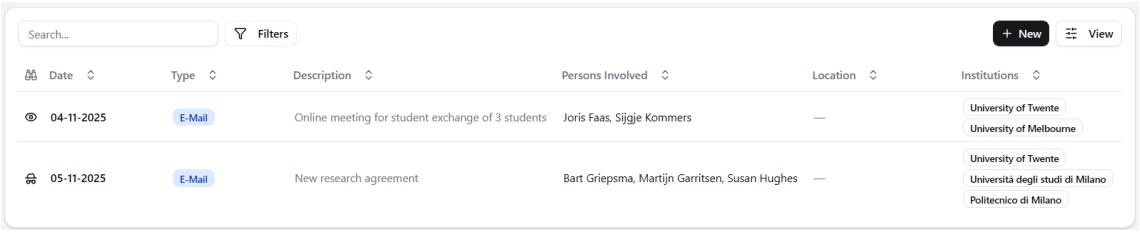


Figure 9

6. Entering information

BIRD allows for entering information in many different places. For example when creating a contact moment it is possible to add a new person to the system. To make this document easier to read, each different concept has their own section. The sections will also explain from where it is possible to reach these “dialogs”.

6.1 Institutions

6.1.1 Viewing

List of all institutions in the system. See 4.3 Tables for more information.

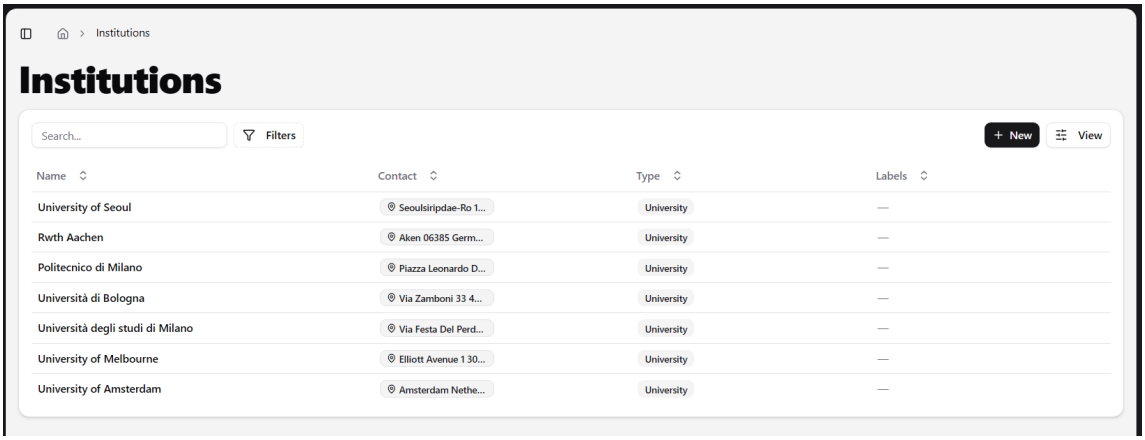


Figure 10

Institution page

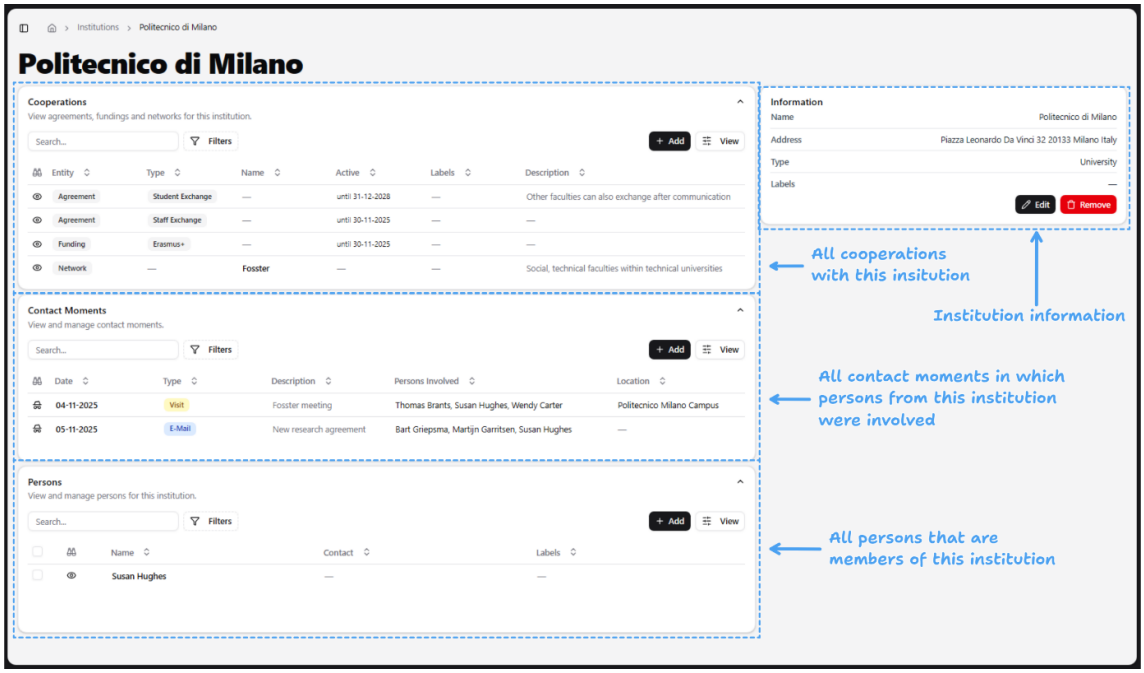


Figure 11

6.1.2 Creation

Accessible from institutions list (on institutions and cooperation pages), institutions selector in side other dialogs.

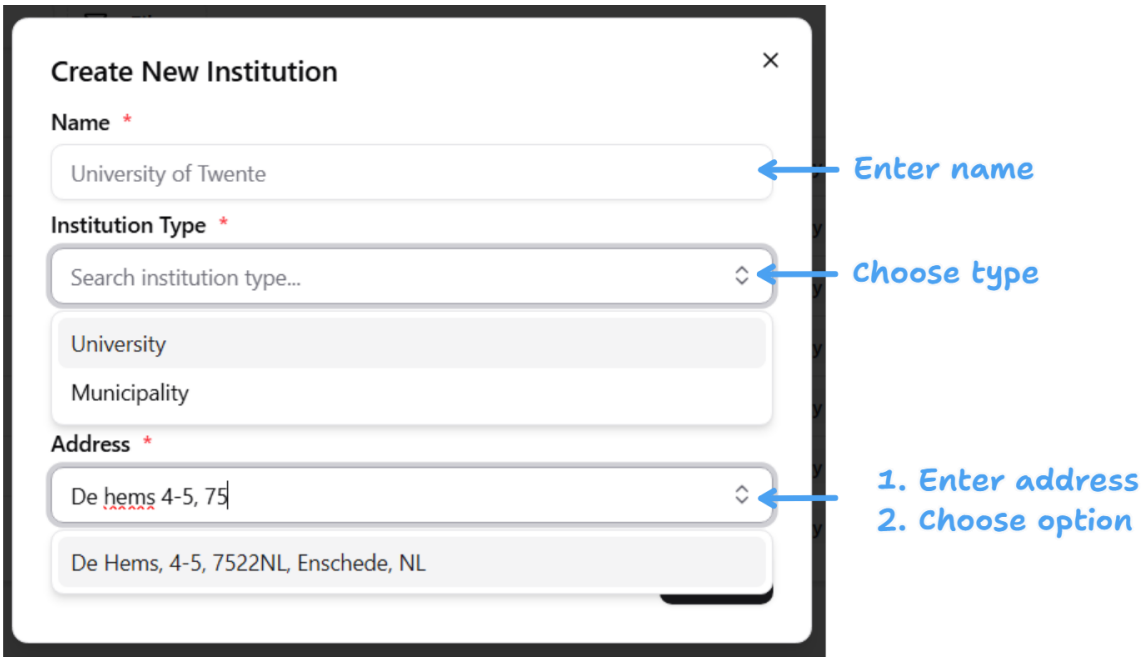


Figure 12

Click on “Create” button to submit.
You will be redirected to the new institution if created from institutions page.

Institution selector

This field can appear in other dialogs. When pressing the “New” button you will open the dialog as shown above.

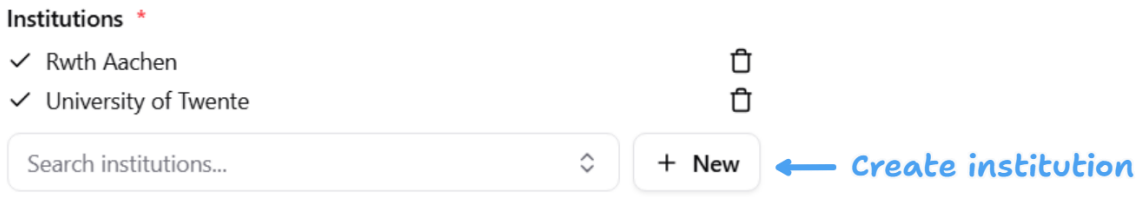


Figure 13

6.2 Persons

6.2.1 Viewing

List of all persons in the system. See 4.3 Tables for more information.

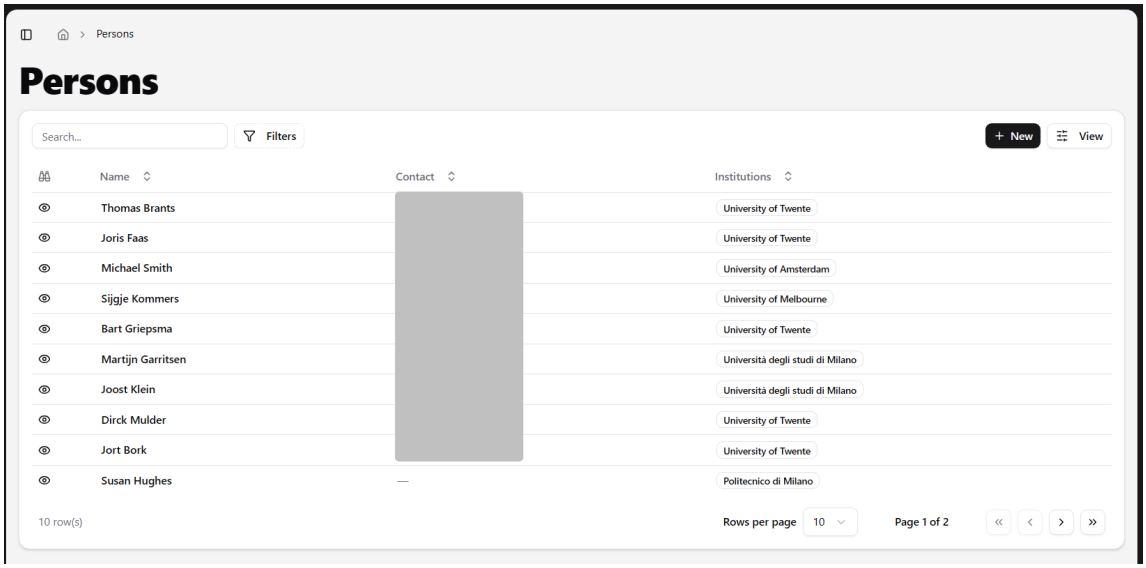


Figure 14

Person page

Figure 15

Person contacts can be seen as “friends” list. This feature is useful to map out relations BMS employees have with employees other institutions.

6.2.2 Creation

Accessible from persons list (on persons, institution and contact moment pages), person selector in side other dialogs.

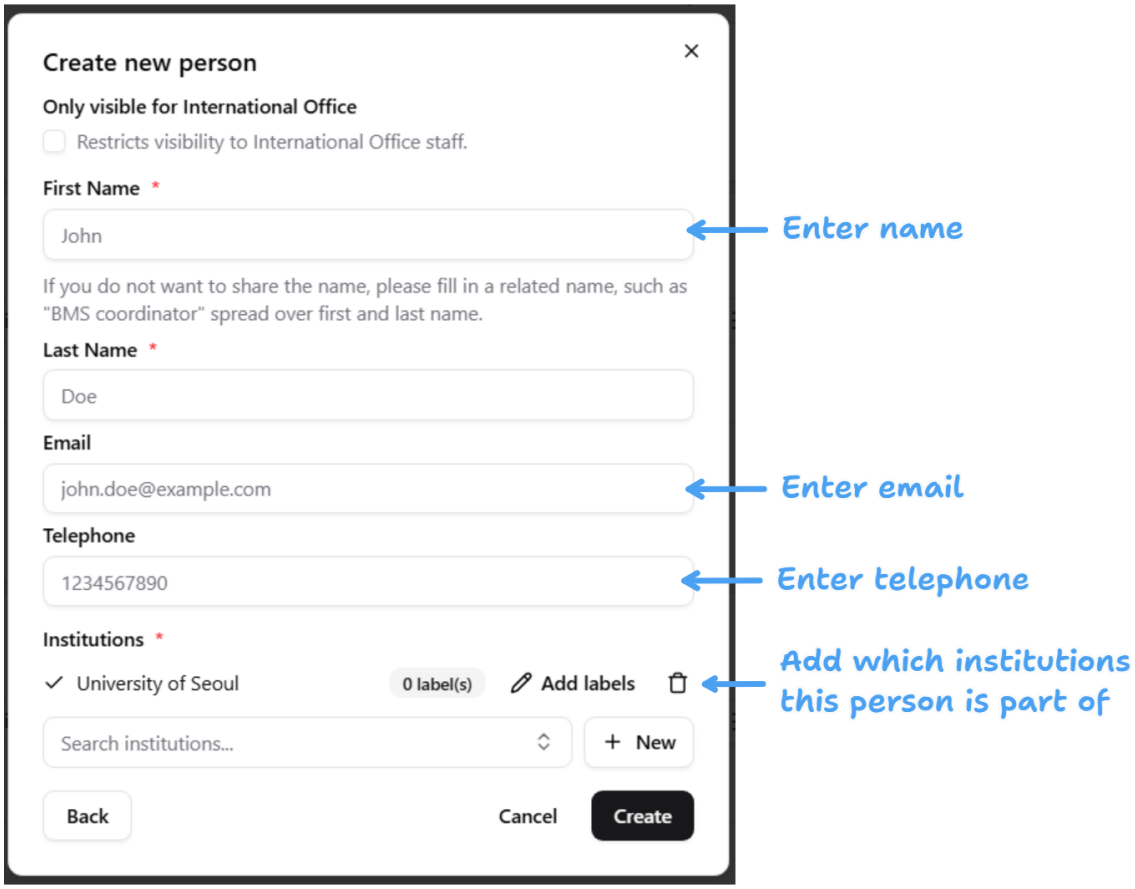


Figure 16

Labels can be added to a person institution connection. This can be used to add more information to this link such as which faculty a person is in.

Click on “Create” button to submit.
You will be redirected to the new person if created from persons page.

Person selector

This field can appear in other dialogs. When pressing the “New” button you will open the dialog as shown above.

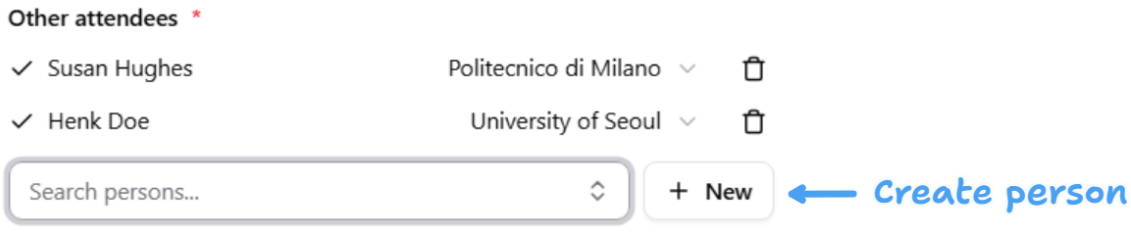


Figure 17

6.3 Cooperations

Cooperations can be created from both their individual lists and from an institution.

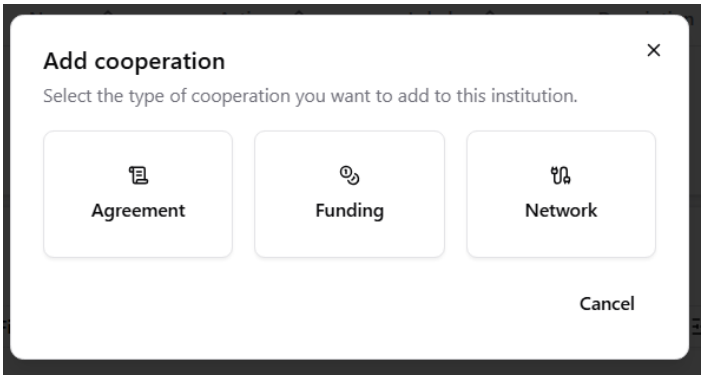


Figure 18

When adding a cooperation to a institute the option is given to either create a new cooperation or add an existing one.

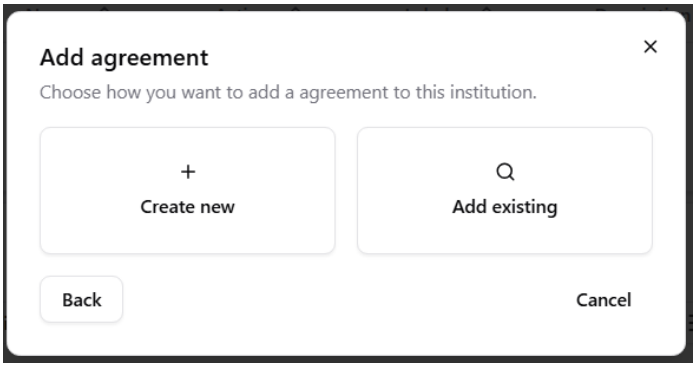


Figure 19

6.3.1 Agreements

6.3.1.1 Viewing

List of all agreements in the system. See 4.3 Tables for more information.

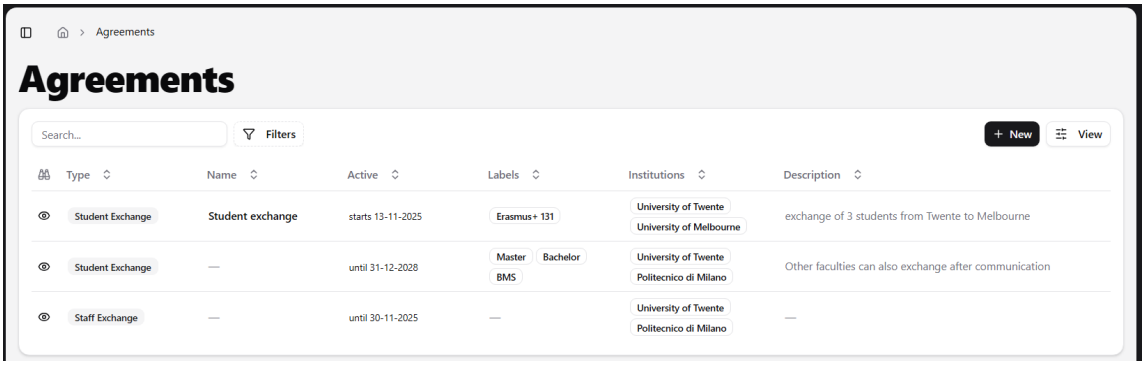


Figure 20

Agreement page

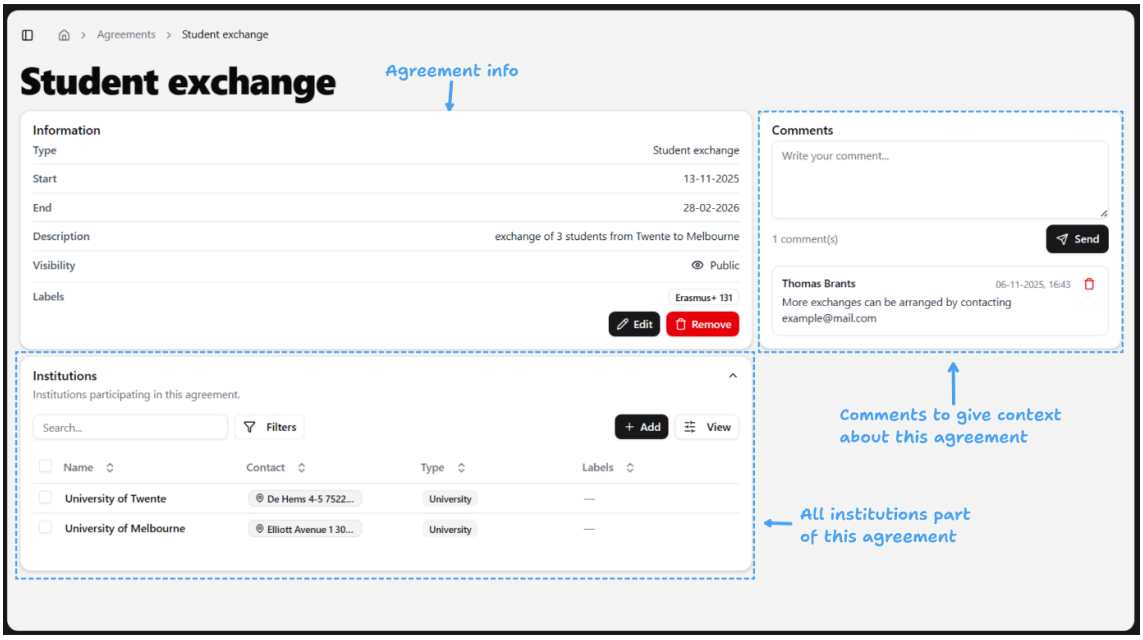


Figure 21

6.3.1.2 Creation

Accessible from see 6.3 Cooperations.

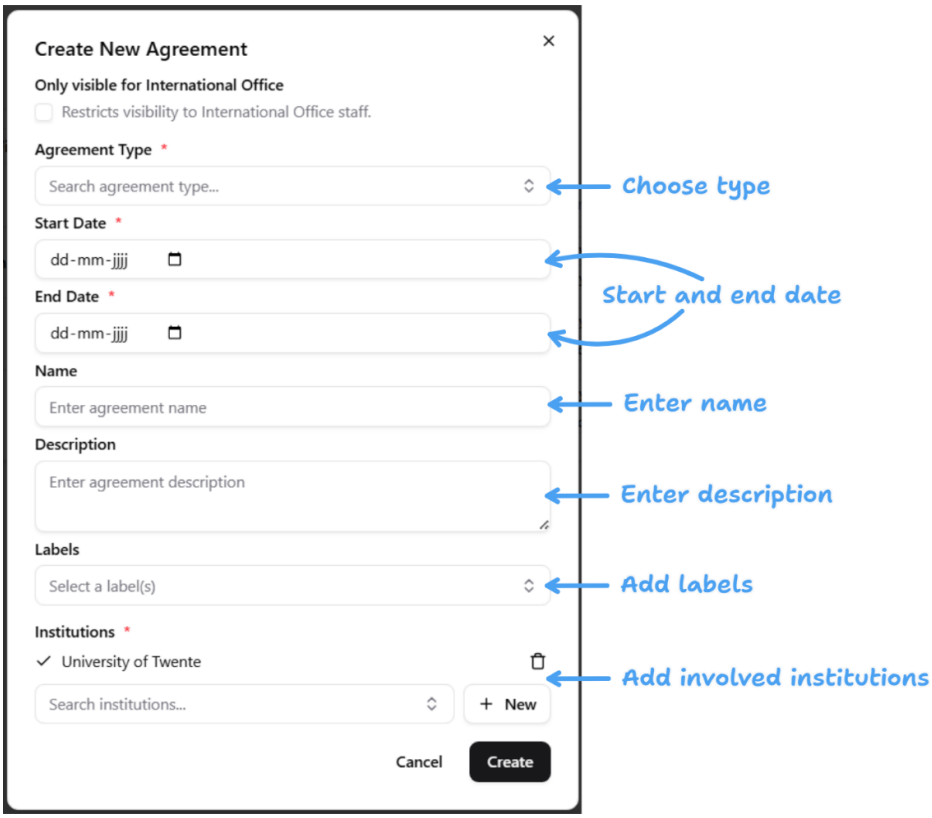


Figure 22

Labels can be used to add more information such as “Study Level”.

Click on “Create” button to submit.
You will be redirected to the new agreement if created from agreements page.

6.3.2 Networks

6.3.2.1 Viewing

Similar to agreements see 6.3.1.1 Agreements.

6.3.2.2 Creation

Accessible from see 6.3 Cooperations.

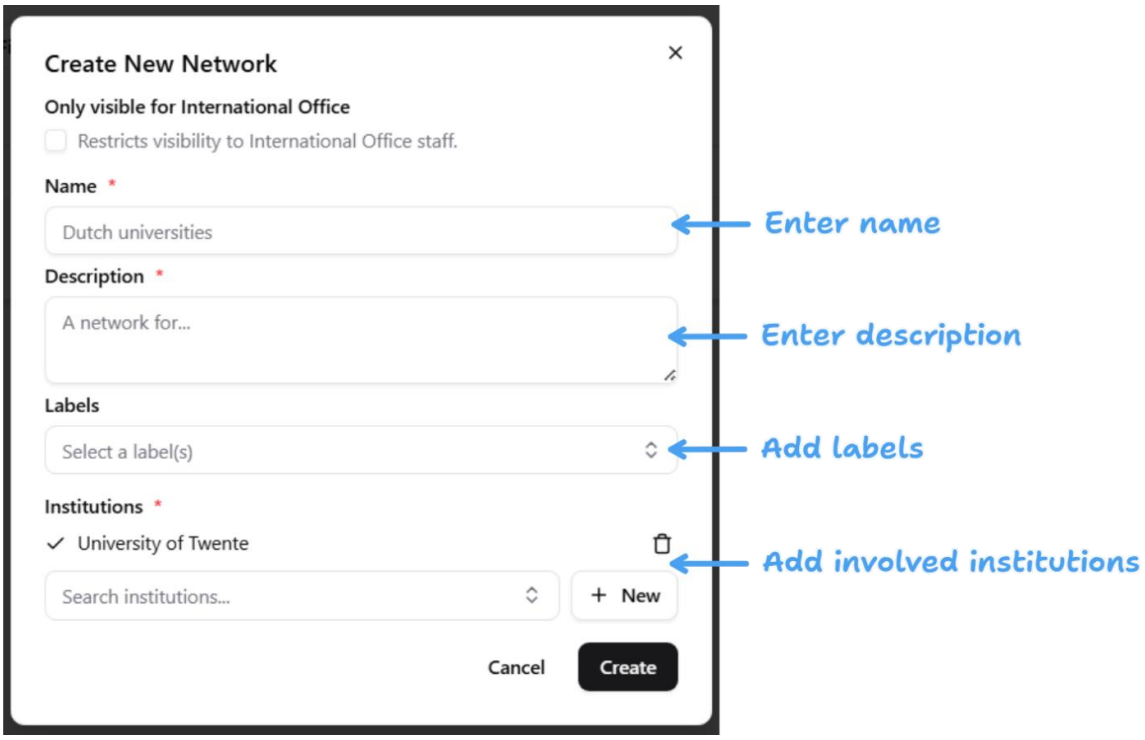


Figure 23

Labels can be used to add more information such as research objectives.

Click on “Create” button to submit.
You will be redirected to the new network if created from networks page.

6.3.3 Funding

6.3.3.1 Viewing

Similar to agreements see 6.3.1.1 Agreements.

6.3.3.2 Creation

Accessible from see 6.3 Cooperations.

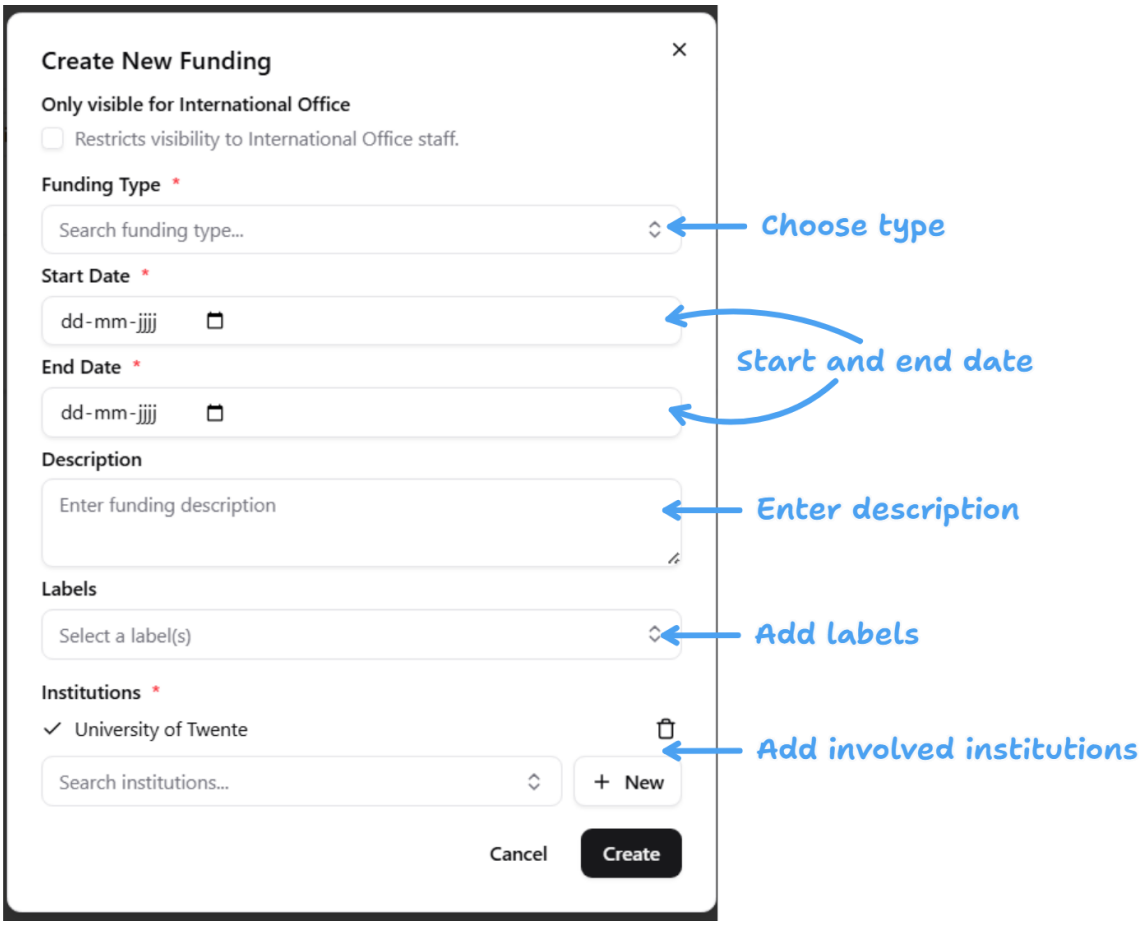


Figure 24

Click on “Create” button to submit.
You will be redirected to the new funding if created from fundings page.

6.4 Contact moments

6.4.1 Viewing

List of all contact moments in the system. See 4.3 Tables for more information.

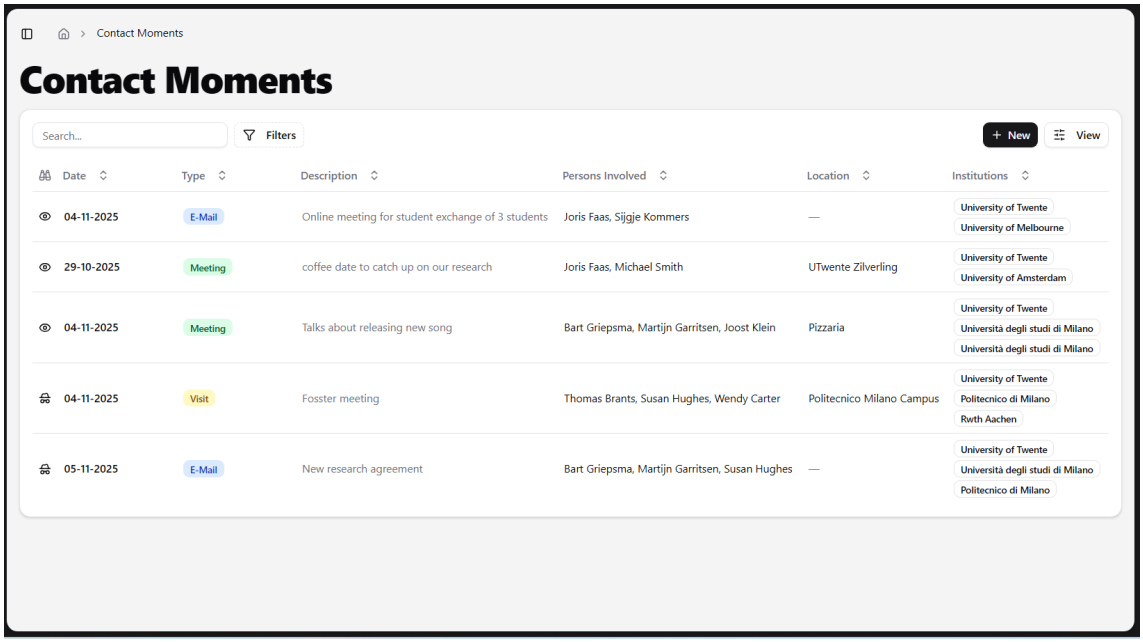


Figure 25

Contact moment page

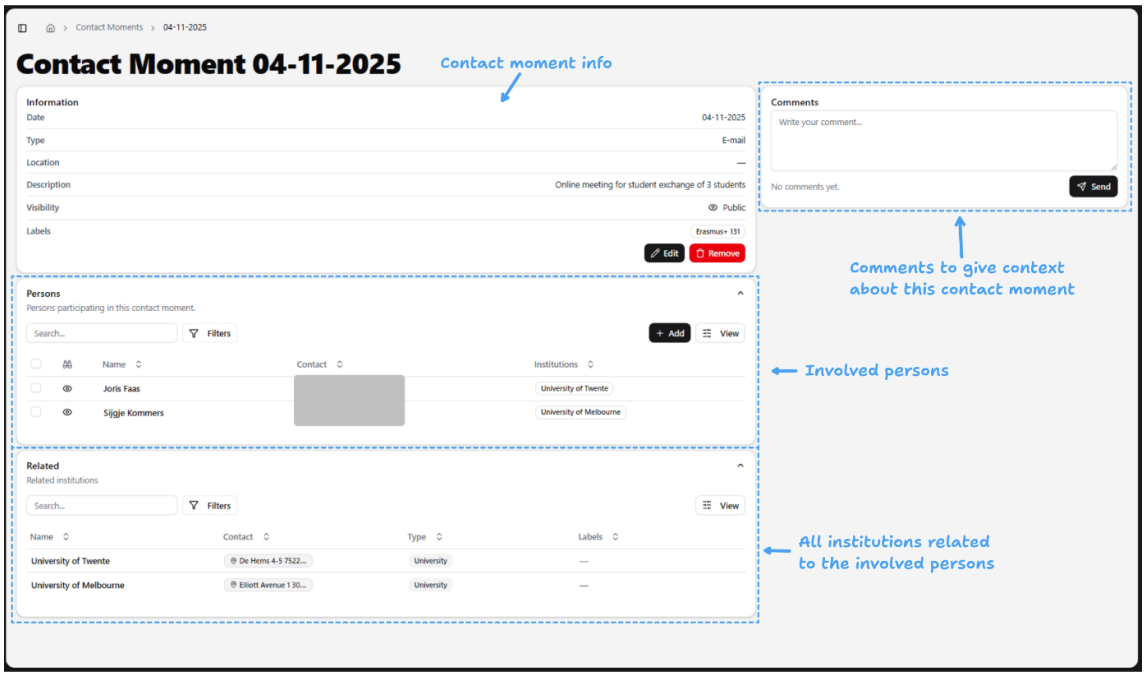


Figure 26

6.4.2 Creation

Accessible from contact moments list (on contact moments, institution and person pages).

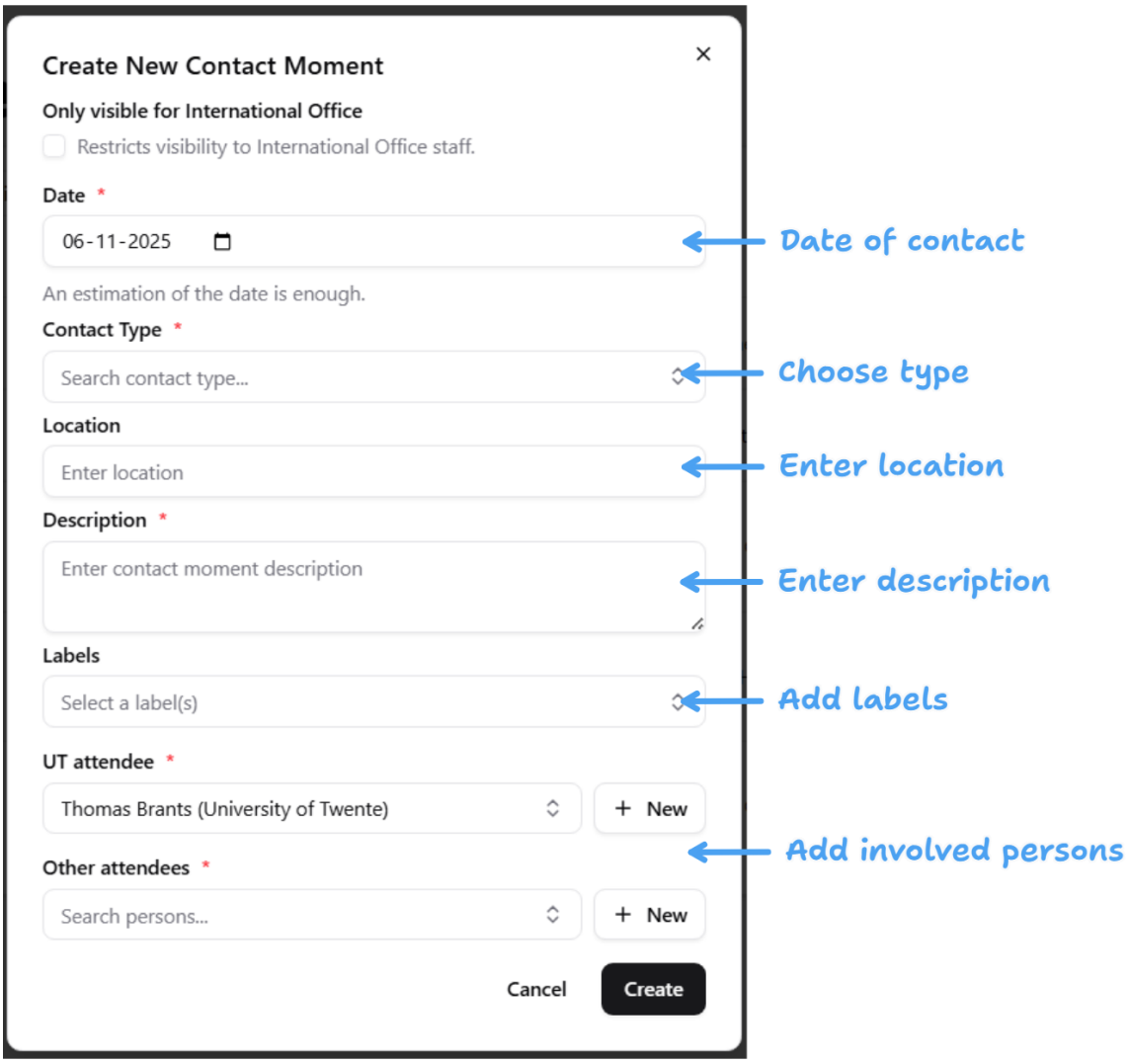


Figure 27

Click on “Create” button to submit.
You will be redirected to the new contact moment if created from contact moments page.

6.5 Settings

The settings are only accessible for Staff users. Here labels and types can be created, edited and removed. The settings page has two tabs: labels and types.

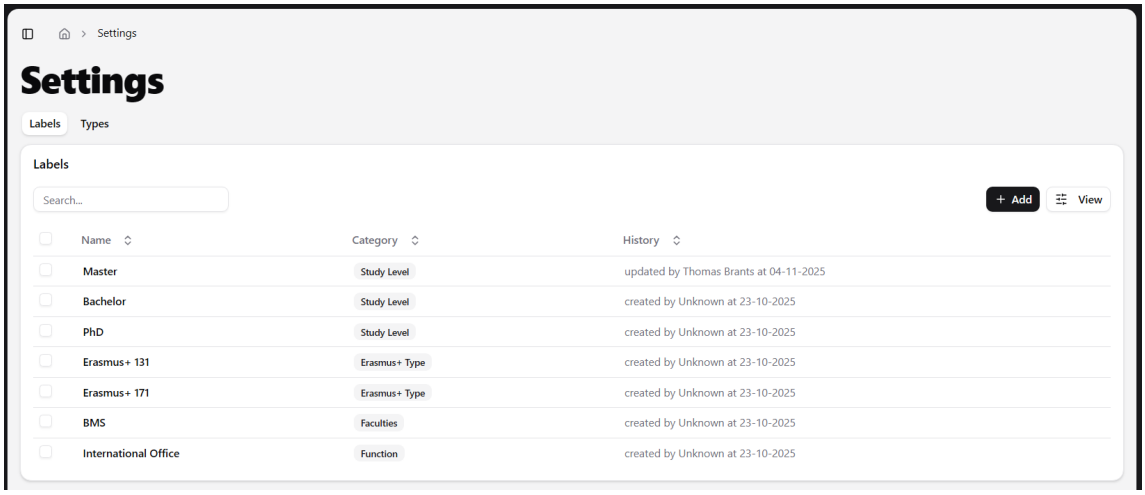


Figure 28

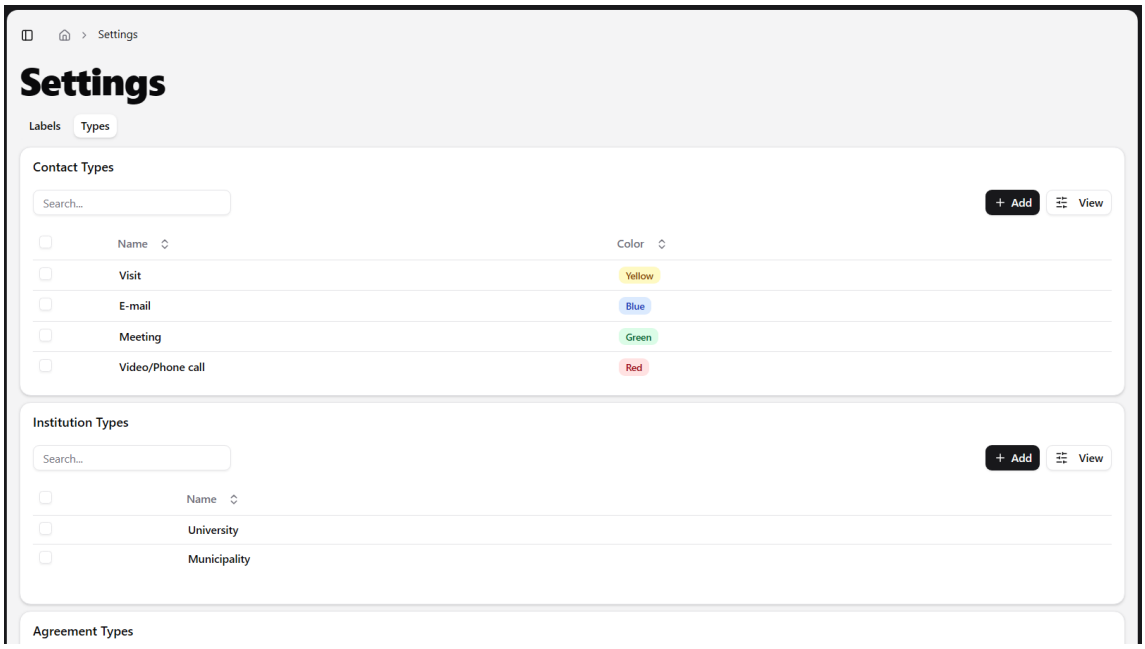


Figure 29

6.5.1 Creation of labels and types

The labels and types creation dialogs are simple, they only contain the text visible and sometimes some extra information such as label category.

Figure 30

7. Credits

Initial concept

BMS International Office, with special thanks to Eline de Ruiter

Supervisor and maintainer

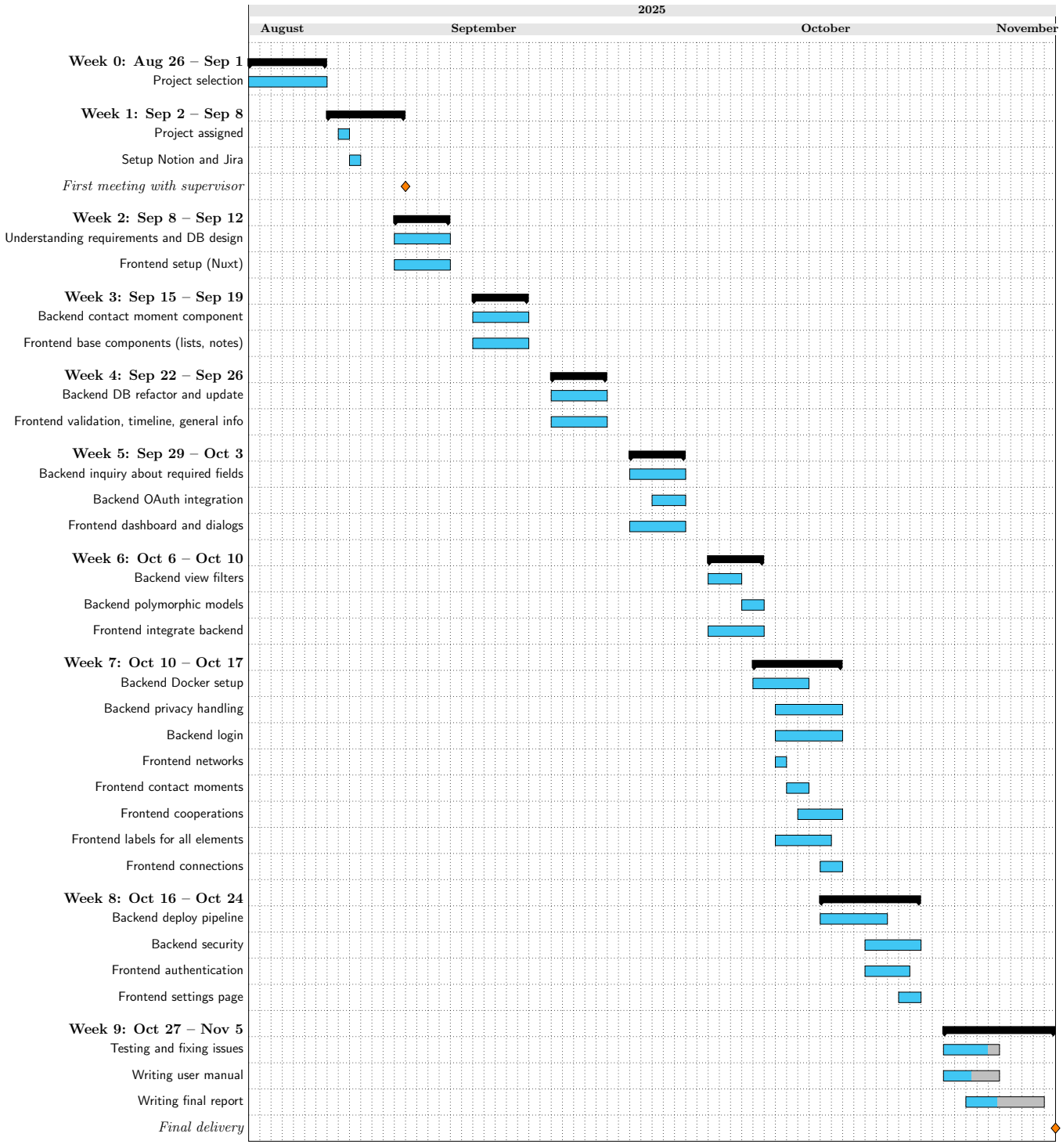
David Huistra

Designed and created by

- Bart Griepsma
- Dirck Mulder
- Jelmer Otten
- Joris Faas
- Jort Bork
- Thomas Brants

Appendix B

Gantt chart


2025
August
September
October
November
Week 0: Aug 26 – Sep 1
Project selection
Week 1: Sep 2 – Sep 8
Project assigned
Setup Notion and Jira
First meeting with supervisor
Week 2: Sep 8 – Sep 12
Understanding requirements and DB design
Frontend setup (Nuxt)
Week 3: Sep 15 – Sep 19
Backend contact moment component
Frontend base components (lists, notes)
Week 4: Sep 22 – Sep 26
Backend DB refactor and update
Frontend validation, timeline, general info
Week 5: Sep 29 – Oct 3
Backend inquiry about required fields
Backend OAuth integration
Frontend dashboard and dialogs
Week 6: Oct 6 – Oct 10
Backend view filters
Backend polymorphic models
Frontend integrate backend
Week 7: Oct 10 – Oct 17
Backend Docker setup
Backend privacy handling
Backend login
Frontend networks
Frontend contact moments
Frontend cooperations
Frontend labels for all elements
Frontend connections
Week 8: Oct 16 – Oct 24
Backend deploy pipeline
Backend security
Frontend authentication
Frontend settings page
Week 9: Oct 27 – Nov 5
Testing and fixing issues
Writing user manual
Writing final report
Final delivery

Appendix C

API Endpoints

Institutions:
Used to manage partner institutions, such as universities and organizations.
Endpoints:
GET /api/institutions/ – Retrieve all institutions.
GET /api/institutions/{id}/ – Retrieve a specific institution.
POST /api/institutions/ – Create a new institution.
PUT/PATCH /api/institutions/{id}/ – Update an existing institution.
DELETE /api/institutions/{id}/ – Delete an institution.
GET /api/institutions/ut/ – Retrieve the institution marked as UT.

Institution Types:
Defines the category of institutions (e.g., University).
Endpoints:
GET /api/institutions/types/ – List all institution types.
GET /api/institutions/types/{id}/ – Retrieve a single institution type.
POST /api/institutions/types/ – Create a new institution type.
PUT/PATCH /api/institutions/types/{id}/ – Update an institution type.
DELETE /api/institutions/types/{id}/ – Delete an institution type.

Addresses:
Stores location and contact address details linked to institutions.
Endpoints:
GET /api/addresses/ – Retrieve all addresses.
GET /api/addresses/{id}/ – Retrieve one address.

Countries:

Provides a list of countries with alpha-2 codes used for address data.
Endpoints:
GET /api/countries/ – Retrieve all countries.

Persons:
Manages contact and staff data, including names, emails, and telephone numbers.
Endpoints:
GET /api/persons/ – Retrieve all persons.
GET /api/persons/{id}/ – Retrieve one person.
POST /api/persons/ – Create a new person.
PUT/PATCH /api/persons/{id}/ – Update person information.
DELETE /api/persons/{id}/ – Delete a person.

Person-Institution Connections:
Represents affiliations between persons and institutions.
Endpoints:
GET /api/persons/institution-connections/ – Retrieve all connections.
GET /api/persons/institution-connections/{id}/ – Retrieve one connection.
POST /api/persons/institution-connections/ – Create a new connection.
PUT/PATCH /api/persons/institution-connections/{id}/ – Update a connection.
DELETE /api/persons/institution-connections/{id}/ – Delete a connection.

Person Connections:
Captures relationships or collaborations between two persons.
Endpoints:
GET /api/persons/persons-connections/ – Retrieve all person connections.
GET /api/persons/persons-connections/{id}/ – Retrieve a specific connection.
POST /api/persons/persons-connections/ – Create a new connection.
PUT/PATCH /api/persons/persons-connections/{id}/ – Update a connection.
DELETE /api/persons/persons-connections/{id}/ – Delete a connection.

Contact Moments:
Logs interactions (meetings, calls, emails) between persons and institutions.
Endpoints:
GET /api/contactmoments/ – Retrieve all contact moments.
GET /api/contactmoments/{id}/ – Retrieve one contact moment.
POST /api/contactmoments/ – Create a new contact moment.
PUT/PATCH /api/contactmoments/{id}/ – Update an existing record.
DELETE /api/contactmoments/{id}/ – Delete a contact moment.

Contact Moment Types:

Defines types of contact (e.g., E-mail, Visit, Meeting).
Endpoints:
GET /api/contactmoments/types/ – Retrieve all types.
GET /api/contactmoments/types/{id}/ – Retrieve a single type.
POST /api/contactmoments/types/ – Create a new contact moment type.
PUT/PATCH /api/contactmoments/types/{id}/ – Update a contact moment type.
DELETE /api/contactmoments/types/{id}/ – Delete a contact moment type.

Agreements:
Manages agreements or partnerships between institutions.
Endpoints:
GET /api/agreements/ – Retrieve all agreements.
GET /api/agreements/{id}/ – Retrieve one agreement.
POST /api/agreements/ – Create a new agreement.
PUT/PATCH /api/agreements/{id}/ – Update agreement information.
DELETE /api/agreements/{id}/ – Delete an agreement.

Agreement Types:
Defines categories of agreements.
Endpoints:
GET /api/agreements/types/ – List all agreement types.
POST /api/agreements/types/ – Create a new agreement type.
PUT/PATCH /api/agreements/types/{id}/ – Update an agreement type.
DELETE /api/agreements/types/{id}/ – Delete an agreement type.

Fundings:
Handles data related to financial funding and grants.
Endpoints:
GET /api/fundings/ – Retrieve all funding records.
GET /api/fundings/{id}/ – Retrieve one funding.
POST /api/fundings/ – Create a new funding record.
PUT/PATCH /api/fundings/{id}/ – Update a funding.
DELETE /api/fundings/{id}/ – Delete a funding record.

Funding Types:
Defines different funding schemes or programs.
Endpoints:
GET /api/fundings/types/ – Retrieve all funding types.
POST /api/fundings/types/ – Create a new funding type.
PUT/PATCH /api/fundings/types/{id}/ – Update a funding type.
DELETE /api/fundings/types/{id}/ – Delete a funding type.

Networks:
Represents collaborative networks involving multiple institutions.
Endpoints:
GET /api/networks/ – Retrieve all networks.
GET /api/networks/{id}/ – Retrieve one network.
POST /api/networks/ – Create a new network.
PUT/PATCH /api/networks/{id}/ – Update a network.
DELETE /api/networks/{id}/ – Delete a network.

Labels:
Provides a system for categorizing institutions, persons, or agreements.
Endpoints:
GET /api/labels/ – Retrieve all labels.
POST /api/labels/ – Create a new label.
PUT/PATCH /api/labels/{id}/ – Update a label.
DELETE /api/labels/{id}/ – Delete a label.

Comments:
Allows users to attach comments to objects such as agreements, fundings, networks, and contact moments.
Endpoints:
GET /api/comments/ – Retrieve all comments.
GET /api/comments/{id}/ – Retrieve one comment.
POST /api/comments/ – Create a new comment.
PUT/PATCH /api/comments/{id}/ – Update a comment.
DELETE /api/comments/{id}/ – Delete a comment.
POST /api/contactmoments/{id}/comments/ – Add a comment to a contact moment.
POST /api/agreements/{id}/comments/ – Add a comment to an agreement.
POST /api/fundings/{id}/comments/ – Add a comment to a funding.
POST /api/networks/{id}/comments/ – Add a comment to a network.

Import JOIN:
Used to import data from the JOIN system into the local database.
Endpoints:
POST /api/import-join/ – Import data from JOIN.

Import Mobility Online:
Used to import data from the Mobility Online platform.
Endpoints:
POST /api/import-mobility-online/ – Import data from Mobility Online.

Microsoft:
Handles authentication through Microsoft's OAuth 2.0 (Single Sign-On).
Endpoints:
GET /microsoft-login/ – Start Microsoft login, storing redirect path.
GET /microsoft-login/after-login/ – Process successful Microsoft login, issue tokens, and redirect user.
POST /api/token/refresh/ – Refresh access token using a valid refresh token.