BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ADMINISTRATIVE INTERFACE FOR Q SORTING

ADMINISTRATIVNÍ ROZHRANÍ PRO Q ŘAZENÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **NIKITA PASYNKOV**
AUTOR PRÁCE

SUPERVISOR **prof. Ing., ADAM HEROUT, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2025

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

162958

Institut:	Department of Computer Graphics and Multimedia (DCGM)
Student:	**Pasynkov Nikita**
Programme:	Information Technology
Title:	**Administrative interface for Q sorting**
Category:	User Interfaces
Academic year:	2024/25

Assignment:

1. Get familiar with the issues of user interface design and development and UX.
2. Get familiar with the issues of Q sorting and the tools for its digital implementation.
3. Design key elements of the administrative part of the system to support Q sorting. Focus in particular on editing research card files, on managing research administrators and research participants, and on organizing individual research projects and their runs.
4. Prototype sub-elements of the interface, test them, and improve them iteratively.
5. Integrate the designed and implemented elements into the system under development.
6. Evaluate the properties of the created solution and identify its strengths and weaknesses, and the room for further development.
7. Evaluate the achieved results and propose possible future work; create a poster and a short video to present the project.

Literature:

- Elisa Păduraru, Fundamentals of Creating a Great UI/UX, Creative Tim, 2022
- Tidwell et al.: Designing Interfaces: Patterns for Effective Interaction Design, O'Reilly, 2020
- Steve Krug: Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability, ISBN: 978-0321965516
- Steve Krug: Rocket Surgery Made Easy: The Do-It-Yourself Guide to Finding and Fixing Usability, ISBN: 978-0321657299

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Herout Adam, prof. Ing., Ph.D.**
Head of Department:	Černocký Jan, prof. Dr. Ing.
Beginning of work:	1.11.2024
Submission deadline:	14.5.2025
Approval date:	8.5.2025

Abstract

This bachelor thesis describes the design and implementation of an administrative application supporting research based on Q-methodology. The application allows researchers to create and manage studies, prepare and version card sets, organize research rounds, invite participants, and track their progress. A flexible access control system ensures that permissions can be assigned accurately across different users and entities. The system is designed to cooperate with an external participant-facing application, forming a complete workflow from study preparation to data collection. Emphasis is placed on usability, modularity, and support for iterative research. Informal usability testing helped refine the user interface and interaction flow. The result is a secure, extensible, and researcher-friendly platform for conducting Q-methodology studies.

Abstrakt

Tato bakalářská práce popisuje návrh a implementaci administrátorské aplikace podporující výzkum založený na Q-metodologii. Aplikace umožňuje výzkumníkům vytvářet a spravovat studie, připravovat a verzovat sady karet, organizovat výzkumná kola, zvát účastníky a sledovat jejich pokrok. Flexibilní systém oprávnění zajišťuje přesné řízení přístupu mezi jednotlivými uživateli a entitami. Systém je navržen pro spolupráci s externí aplikací pro účastníky, čímž vzniká ucelený proces od přípravy studie po sběr dat. Důraz je kladen na použitelnost, modularitu a podporu iterativního výzkumu. Neformální testování použitelnosti přispělo k vylepšení rozhraní a interakce. Výsledkem je bezpečná, rozšiřitelná a uživatelsky přívětivá platforma pro realizaci výzkumů metodou Q.

Keywords

Q-methodology, web-based research, study management, card sorting, access control, usability, NestJS, PostgreSQL, Next.js, CASL

Klíčová slova

Q-metodologie, online výzkum, správa studií, třídění karet, řízení přístupu, použitelnost, NestJS, PostgreSQL, Next.js, CASL

Reference

PASYNKOV, Nikita. *Administrative interface for Q sorting.* Brno, 2025. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Ing., Adam Herout, Ph.D.

Administrative interface for Q sorting

Declaration

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana prof. Ing. Adama Herouta Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Nikita Pasynkov
May 14, 2025

Contents

Chapter 1

Introduction

Q-methodology is a research approach used to examine human subjectivity, including views, beliefs, and attitudes. In Q-methodology research, participants are given a collection of statements which they have to arrange according to their opinion - if they agree or disagree with the statement. This sorting procedure helps researchers understand trends in individuals' thoughts on a specific issue. Q-methodology is frequently used in social sciences, psychology, healthcare, and education to analyze subjective opinions in more depth than other surveys.

This project aims to create an administrative panel that helps researchers working with the Q-methodology. Although several digital tools are available to help both researchers and participants with Q-methodology surveys, there is frequently inadequate support for administrators tasked with study preparation, participant management, card set editing, and research round progress tracking. Often, these tasks are performed manually or with general-purpose survey instruments that are poorly designed to satisfy the requirements of the Q-methodology.

This project was created to provide a system that enables researchers to handle all phases of a Q-methodology study from a single platform. The admin panel enables users to create and modify studies, schedule rounds, edit card sets, view studies' results, and manage participants. It also includes features such as email notifications, permission control, and versioning of card sets. The goal is to improve the administrative experience by making it more efficient, organized, and user-friendly.

The system is designed with modern tools to achieve this. The backend is built using NestJS, the database is implemented with PostgreSQL. The frontend is built using React with Next.js as the core framework, TailwindCSS and Joy UI for styling, and TanStack Query for communicating with backend. Role-based access control is managed by CASL, allowing users to authenticate with traditional credentials such as JWT tokens or Google OAuth.

The system was designed for flexibility and extensibility, with potential improvements expected based on user feedback.

The application is currently publicly available at `https://qpanel.settler.tech/`.

Chapter 2

About Q-Methodology

British psychologist William Stephenson developed the Q-methodology approach in 1935 [11]. This method of research aims to systematically and quantitatively examine subjectivity. Unlike conventional survey techniques that emphasize the extent to which people agree or disagree with particular statements on a scale, the Q-methodology approach gives the comparison of whole viewpoints top priority. Usually using a quasi-normal distribution, it is usually carried out by having participants rate a set of statements in relation to one another. Called a Q-sort, this ranking method shows how people order or give priority to their opinions on a particular topic.

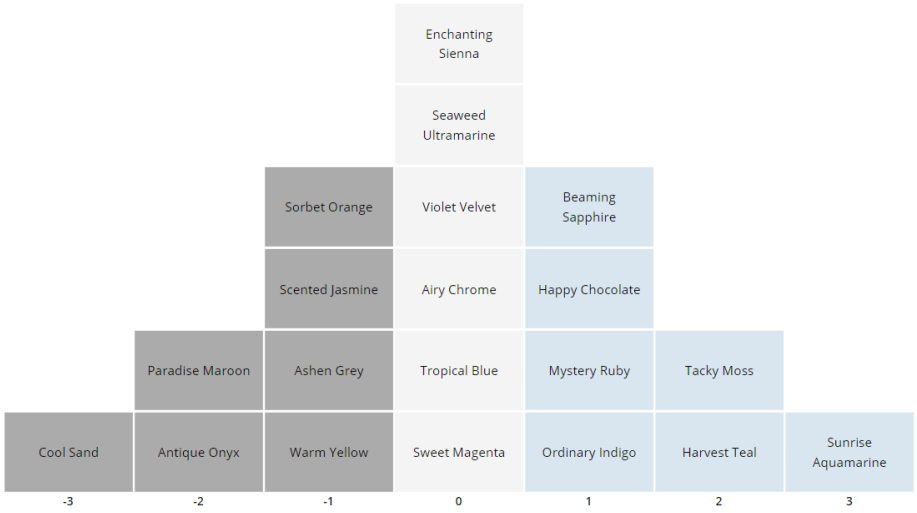

Figure 2.1: An example of a digital Q-sort interface where participants sort statements along a predefined agreement scale.

The main advantages of Q-methodology are its combination of statistical accuracy and qualitative depth, which allows researchers to explore opinions without forcing them into predefined categories [2]. Typically, after collecting classifications, researchers use factor analysis to find clusters of common opinions among participants.

The Q-methodology approach is particularly useful in studies on complex or controversial topics to understand attitudes, values, and opinions. Among the many fields using this approach are psychology, education, health sciences, political science, environmental studies, and organizational research [12]. Examples of researched topics include the evaluation

of patients' views on treatment alternatives, the views of teachers on curriculum changes, or the views of different stakeholders on new company policies.

The method is also praised for its focus on participants. The outcomes usually provide a more complete knowledge of the reasoning behind people's points of view as they organize statements to reflect their own knowledge and judgment. The Q-methodology approach is therefore beneficial for the detection of general trends in attitudes and the generation of large, comprehensible results that could guide design, communication, or policy decisions.

Digital technologies have made it much easier to conduct research online, allowing the method to become more popular in recent years to include a wider spectrum of researchers and participants. Despite this, there are few tools built for researchers who need well-designed administrative tools that support running studies effectively.

Chapter 3

Analysis of Existing Solutions

Although there are a few digital tools for conducting Q-methodology studies, they are often outdated, difficult to use, or poorly optimized for modern research needs. These platforms typically lack support for advanced features such as card set versioning, study structures with iterations, email notifications, collaboration with other researchers, or flexible participant management, which limits their suitability for more complex research goals. This chapter gathers and describes some of the existing applications for conducting research, which shows the need to build a modern, user-friendly application that would suit the needs of researchers.

3.1 QMethodSoftware

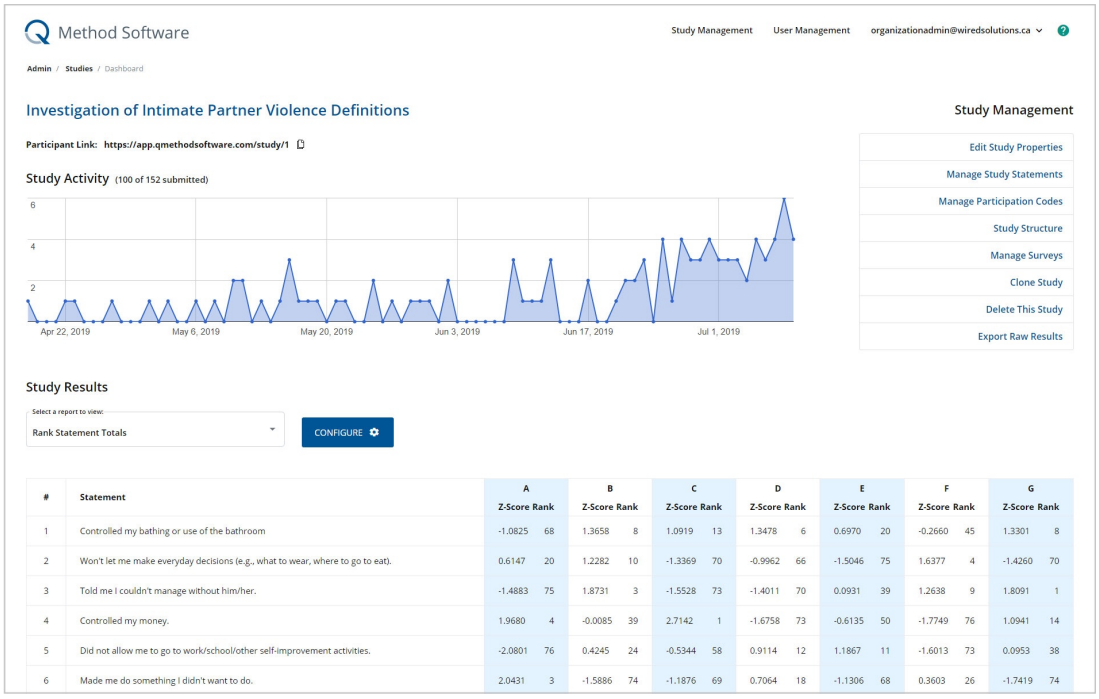

Figure 3.1: QMethodSoftware dashboard with detailed analytics and configuration options.

QMethodSoftware[1] is currently the most feature-rich and technically advanced platform for conducting Q-methodology research. It provides robust support for sorting, multiple media types in statements, real-time analysis, and participant management. However, its pricing model significantly limits its accessibility: many core features necessary for conducting meaningful research, such as email notifications, support for more than one study, more than one survey per study, or more than 10 participants, are only available with the Enterprise plan, which costs $199 per user per month. For students or small research teams, this makes the tool too expensive to use.

Additionally, while the platform offers a wide range of features, the user interface can be overwhelming due to its complexity. There are many nested configuration options, and users may find it difficult to navigate the interface efficiently, especially when setting up iterative studies or customizing participant flows. This complexity can slow down the workflow and make the platform less approachable for researchers new to Q-methodology or digital research tools.

3.2 VQMethod

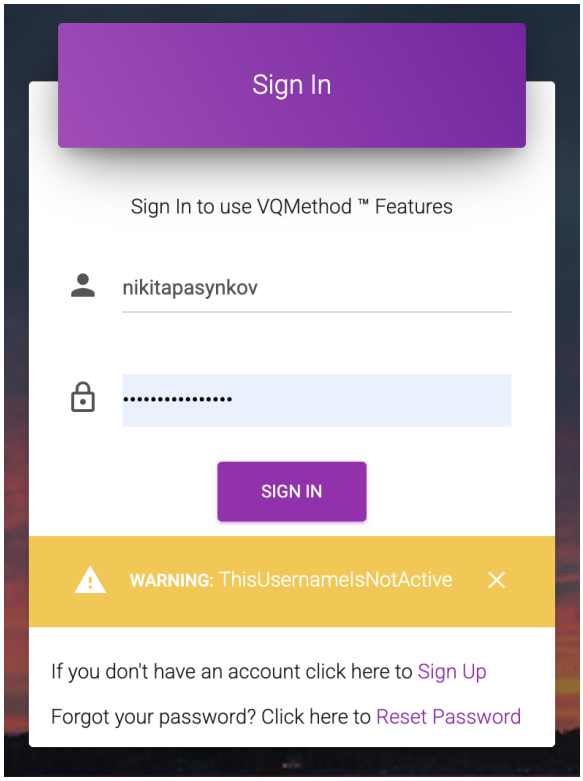

Figure 3.2: VQMethod sign-in screen.

Although VQMethod[2] was considered as a potential alternative, repeated failures of the platform to register made it impossible to test.

[1] https://qmethodsoftware.com
[2] https://vqmethod.com/

3.3 Q-TIP

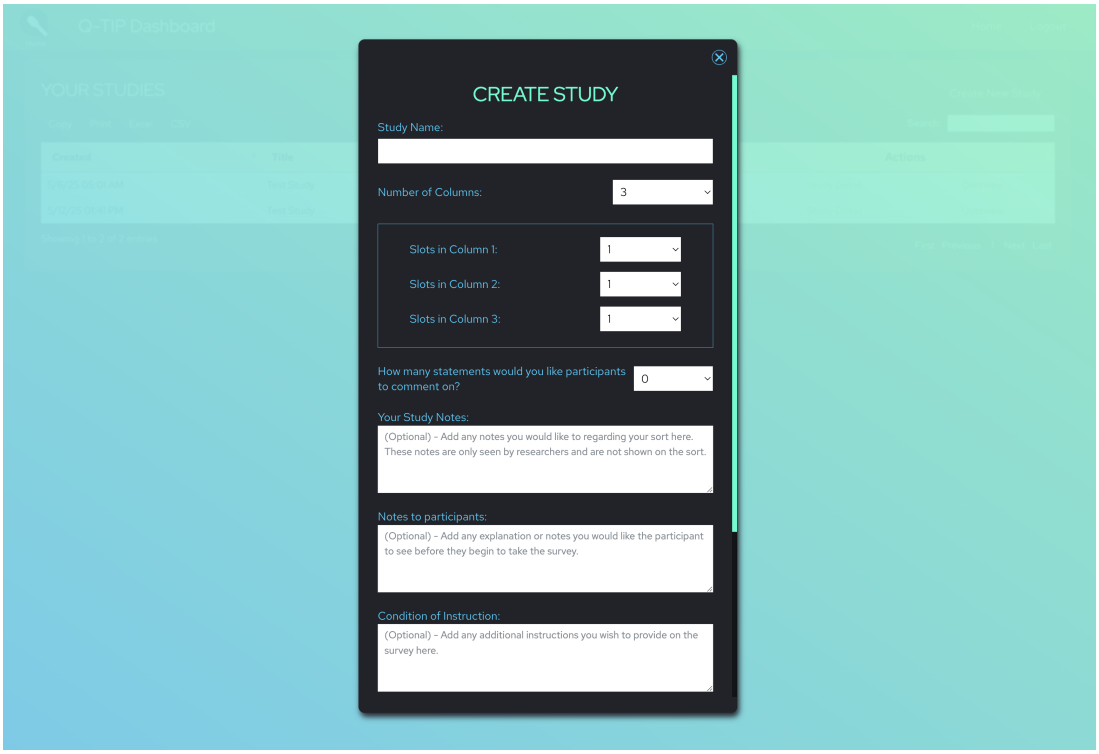

Figure 3.3: Q-TIP user interface.

A free and accessible tool, Q-TIP[3] was developed by the University of Wisconsin. Researchers are able to develop studies and analyze participant responses. A useful feature that helps with the research is the ability to observe participant sessions. However, the user interface may occasionally be confusing and inconsistent. The unclear layout and the fact that all settings are stacked in one long vertical scroll may feel overwhelming to users, as shown in Figure 3.3. The used color palette, especially the excessive bright colors, could discourage the user from engaging with the application. This tool also does not provide email support, which means that researchers have to manually invite participants using a code generated in the application.

3.4 HTMLQ

Another option is HTMLQ[4], an open-source tool available on GitHub. Although it provides a basic framework for conducting Q-methodology studies in the browser, it requires a fair amount of technical knowledge to install, configure, and use. Furthermore, the project has not been actively maintained - the last update was published in 2015 - which may lead to compatibility issues with modern systems or browsers.

[3] `http://qtip.geography.wisc.edu`
[4] `https://github.com/aproxima/htmlq`

3.5 Executive Summary

The most comprehensive solution currently available is QMethodSoftware; however, it has limitations, especially regarding its cost and complexity.

This project presents a modern and flexible administrator panel designed to address the problems found in existing Q-methodology software. The main goal of the application is to be user-friendly, free to use, and easy to customize. It includes features such as managing roles and permissions, sending email notifications, and organizing studies into rounds. It also allows users to create study-independent card sets, which is not supported in existing tools. In general, the application aims to give researchers a better tool to manage their studies and to support more advanced research processes.

Chapter 4

Used Technologies

This chapter describes the key technologies used in the development of the Q-methodology administrative panel, explaining the reasoning behind each choice, and providing code examples.

4.1 Backend Technologies

4.1.1 NestJS

NestJS[1] is a progressive Node.js framework for building efficient and scalable server-side applications. It uses TypeScript by default and combines elements of object-oriented programming, functional programming, and functional reactive programming.

NestJS was chosen for this project because of its modular architecture, which aligns well with complex domain requirements. The framework provides a robust structure for organizing code, enforcing separation of concerns, and implementing clean architecture principles. In addition, its extensive ecosystem of integrations simplifies the connection with databases, authentication providers, and other services.

```
@Module({
  imports: [TypeOrmModule.forRoot({
    type: 'postgres',
    host: process.env.POSTGRES_HOST,
    port: 5432,
    username: process.env.POSTGRES_USER,
    password: process.env.POSTGRES_PASSWORD,
    database: process.env.POSTGRES_DB,
    entities: [__dirname + '/**/*.entity{.ts,.js}'],
  }), StudiesModule, CardDecksModule, AuthModule],
})
export class AppModule {}
```

[1] https://nestjs.com/

4.1.2 PostgreSQL with TypeORM

PostgreSQL[2] was selected as the database solution due to its robustness, reliability, and support for complex data relationships. As a mature relational database system, PostgreSQL offers excellent stability and performance while handling the intricate relationships between studies, rounds, card sets, and users.

TypeORM[3] provides an elegant way to work with databases in a TypeScript environment, offering Active Record and Data Mapper patterns [6]. It was chosen for its type-safe approach to database operations and seamless integration with NestJS.

```
@Entity()
export class Round {
  @PrimaryGeneratedColumn('uuid') id: string;
  @Column() name: string;
  @Column({ type: 'text', nullable: true }) description: string;
  @ManyToOne(() => Study, study => study.rounds) study: Study;
  @ManyToOne(() => CardDeckVersion) cardDeckVersion: CardDeckVersion;
  @OneToMany(
    () => RoundParticipant,
    (roundParticipant) => roundParticipant.round,
  )
  roundParticipants: RoundParticipant[];
}
```

Each decorator in this example defines a specific aspect of how the class is mapped to the database:

- `@Entity()`: Marks the class as a database entity. TypeORM will map this class to a database table called `round`.
- `@PrimaryGeneratedColumn('uuid')`: Declares the primary key of the entity. The `'uuid'` strategy is used to generate a unique identifier for each round, which is particularly helpful in distributed systems or environments where sequential IDs may cause conflicts.
- `@Column()`: Maps a standard property to a table column. The `name` column is required, while the `description` column is marked as `nullable: true`, which means that it is optional in the database.
- `@ManyToOne(() => Study, study => study.rounds)`: Establishes a many-to-one relationship with the `Study` entity. This means that many rounds can belong to a single study. The second argument defines the inverse relationship for bidirectional access.
- `@ManyToOne(() => CardDeckVersion)`: Connects a round to a specific version of a card deck. This enables version tracking and historical consistency for the cards used during the round.

[2] https://www.postgresql.org/
[3] https://typeorm.io/

- `@OneToMany(() => RoundParticipant, roundParticipant => roundParticipant.round)`: Defines a one-to-many relationship with the `RoundParticipant` entity, indicating that a single round can have many associated participants.

This object-relational mapping not only simplifies the development of database logic but also enables automated schema synchronization, migration generation, and improved code maintainability. By working with entities like `Round` directly in TypeScript, developers can benefit from compile-time type checking and IDE support when writing business logic and interacting with the database.

4.1.3 CASL for Authorization

CASL[4] is an isomorphic authorization library that restricts the resources that a user can access. It was chosen to handle role-based access control within the application, allowing fine-grained permission management for different user roles.

```
@Injectable()
export class CaslAbilityFactory {
  createForUser(user: User) {
    const { can, cannot, build } = new AbilityBuilder(createMongoAbility);
    if (user.isAdmin) {
      can('manage', 'all');
    } else {
      can('read', 'all');
      can('manage', Study, { ownerId: user.id });
      can('update', Study, { collaborators: { $elemMatch: { id: user.id } } });
    }

    return build();
  }
}
```

This class uses the `AbilityBuilder` provided by CASL to construct a user's ability definition.

- `can('manage', 'all')` grants administrators full access to all resources.
- `can('read', 'all')` allows all users to read any entity.
- `can('manage', Study, { ownerId: user.id })` ensures that users can only manage the studies they created.
- `can('update', Study, { collaborators: { $elemMatch: { id: user.id } } })` allows users to update studies where they are listed as collaborators.

[4] `https://casl.js.org/v6/en/`

4.2 Frontend Technologies

4.2.1 Next.js

Next.js[5] is a React[6] framework that enables server-side rendering, static site generation, and other advanced features with minimal configuration. It was chosen for the frontend due to its excellent developer experience, built-in routing system, and optimized performance.

```
export default function StudyPage() {
  const router = useRouter();
  const { id } = router.query;
  const { data: study, isLoading, error } = useQuery({
    queryKey: ['study', id],
    queryFn: () => fetchStudyById(id as string),
    enabled: !!id,
  });
  if (isLoading) return;
  if (error) return;
  return;
}
```

4.2.2 TanStack Query

TanStack Query[7] is a data-fetching and state management library that simplifies server state handling in React applications. It was selected for its powerful caching, background updates, and UI capabilities.

```
export function useStudies() {
  const queryClient = useQueryClient();
  const studies = useQuery({ queryKey: ['studies'], queryFn: fetchStudies });
  const updateMutation = useMutation({
    mutationFn: (updatedStudy: Study) => updateStudy(updatedStudy),
    onSuccess: (data) => {
      queryClient.invalidateQueries({ queryKey: ['studies'] });
      queryClient.invalidateQueries({ queryKey: ['study', data.id] });
    },
  });
  return {
    updateStudy: updateMutation.mutate,
  };
}
```

[5] https://nextjs.org/
[6] https://react.dev/
[7] https://tanstack.com/query/v3/

4.2.3 MUI Joy and TailwindCSS

MUI Joy[8] is a component library focused on providing a modern, clean design system. Combined with TailwindCSS[9], a utility-first CSS framework, this pairing offers both pre-built components and the flexibility to create custom designs efficiently.

```
<Button
  type="submit"
  loading={isSubmitting}
  variant="solid"
  className="w-full bg-primary-600 hover:bg-primary-700">
  {initialData?.id ? 'Update' : 'Create'}
</Button>
```

4.3 Authentication Solutions

4.3.1 JWT with Passport.js

JSON Web Tokens (JWT) combined with Passport.js[10] provide a robust authentication strategy for the application. This approach was selected for its stateless nature, which simplifies scaling and deployment, while Passport's extensible architecture supports multiple authentication methods [8].

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  async validate(payload: any) {
    const user = await this.usersService.findOne(payload.sub);
    return user;
  }
}
```

4.3.2 Google OAuth Integration

Google OAuth was implemented to provide users with a simplified login experience [4], reducing friction during account creation and login. This approach enhances security by delegating authentication to Google's trusted infrastructure while simplifying the authentication process.

[8] https://mui.com/joy-ui/getting-started/
[9] https://tailwindcss.com/
[10] https://www.passportjs.org/

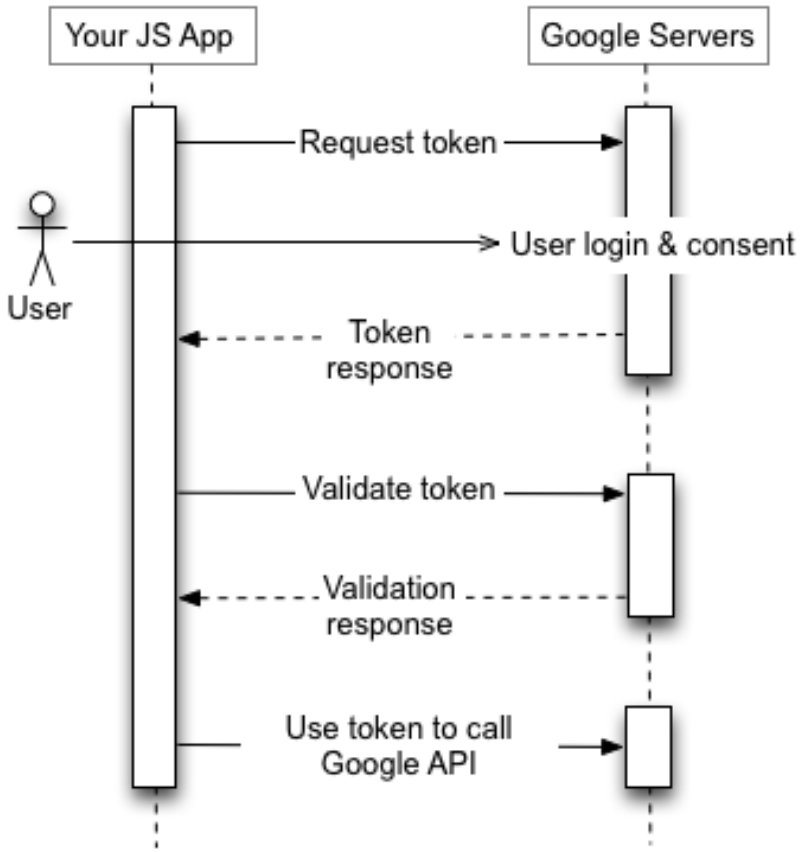

Figure 4.1: Google OAuth 2.0 authentication flow, illustrating how authorization codes and tokens are exchanged between the application and Google servers.

4.4 Conclusion

The technology stack selected for this project represents a balance between developer productivity, application performance, and maintainability. NestJS and TypeORM provide a structured and type-safe backend foundation, while Next.js and TanStack Query enable building a responsive and efficient frontend experience. Security is addressed through integrated solutions like CASL for authorization and Passport.js with JWT and Google OAuth for authentication.

Each technology choice was made considering the specific requirements of a Q-methodology administration system, focusing on modularity, extensibility, and user experience. This modern stack enables the creation of a robust application that addresses the limitations found in existing Q-methodology tools while providing a platform that can evolve to meet future research needs.

Chapter 5

Application Design

5.1 Use Case Diagram

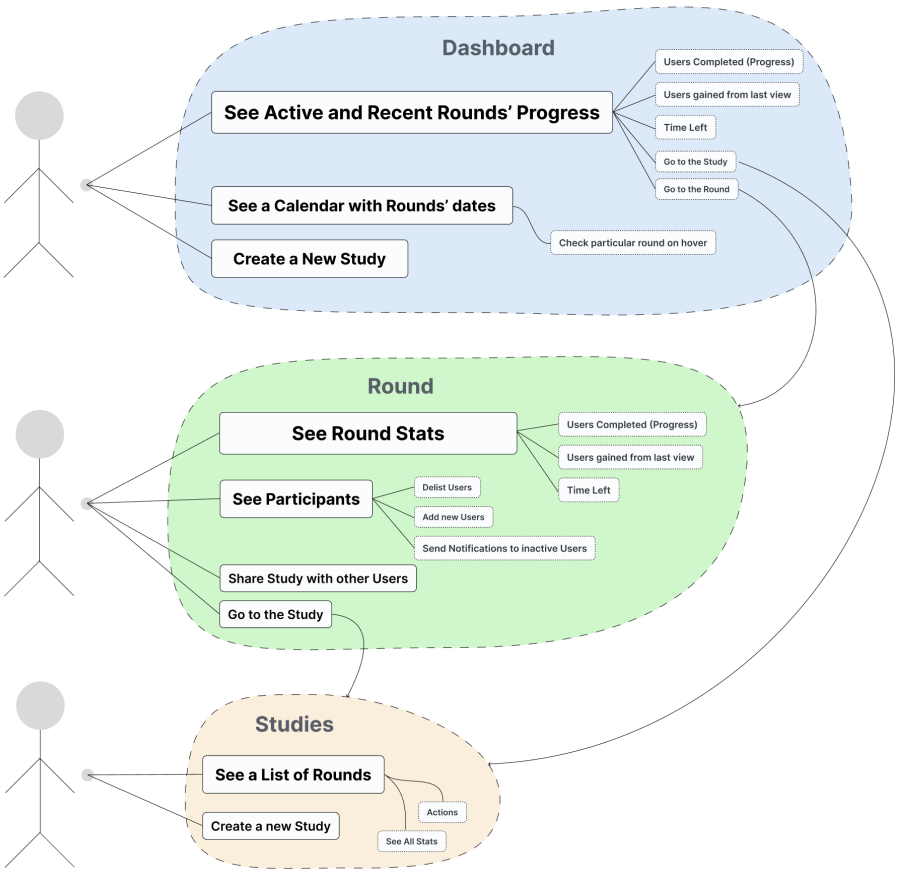

Figure 5.1: Use case diagram outlining administrator interactions across Dashboard, Round, and Study sections.

Before designing the database model, a use case diagram was created to map out the essential features and user interactions of the application. This step helped define the core functionality from the user's perspective and guided the overall system architecture. The

diagram highlights the main parts of the system - Dashboard, Round, and Studies - and outlines the typical actions an administrator can perform in each area. These include creating studies, managing participants, tracking progress, sending notifications, and sharing access with other users. This high-level overview served as the foundation for the data model and the application logic that followed.

5.2 Entity-Relationship Diagram

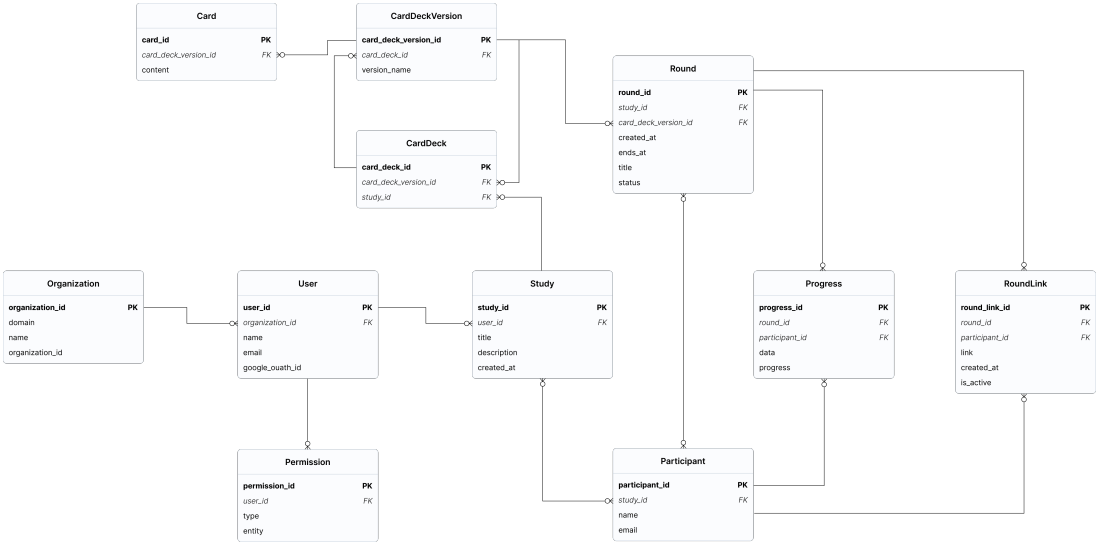

Figure 5.2: Entity-Relationship Diagram (ERD) representing the core data model of the application.

This chapter presents the design of the application and describes the process of building it. At first, an abstract data model was created to reflect the structure and relationships between key entities of the system. This model served as the foundation for the implementation of the backend using modern web technologies. After defining the initial schema, the development continued with the implementation of the core logic and user interface. Several iterations followed, during which the data model, business logic, and user interface were gradually adjusted and refined based on practical feedback and testing.

The core of the system is built around several key entities that represent the main objects in a Q-methodology research workflow: studies, rounds, participants, users, cards, and permissions. The ERD also includes supporting structures such as card deck versions and organizations, enabling scalability and collaboration across institutions.

5.3 Entity Descriptions

5.3.1 Study

Each study represents a research project managed by a user. It includes attributes such as:

- study_id – primary identifier
- user_id – foreign key to the study's creator

- title, description – textual metadata
- created_at – timestamp of study creation

A study can have multiple rounds and participants and may be linked to a card deck.

5.3.2 Round

The Round entity represents a specific iteration of a survey within a study. This allows researchers to conduct multiple rounds (e.g. pilot testing, final study) using the same or updated card set. Attributes include:

- round_id – primary identifier
- study_id – foreign key to the parent study
- card_deck_version_id – links to the specific version of cards used
- created_at, ends_at – timestamps for scheduling
- title, status – metadata for UI and logic

5.3.3 CardDeck, CardDeckVersion, Card

The card-related entities allow modular management of statement sets:

- **CardDeck** is the high-level container for a group of statements. It is not tied to a specific study round and can be reused.
- **CardDeckVersion** allows versioning of a deck, allowing administrators to update cards without losing historical context.
- **Card** includes:
  - card_id – primary key
  - card_deck_version_id – links to the version it belongs to
  - content – the actual text of the statement

This versioning design improves traceability and reuse.

5.3.4 Participant

A participant is an individual invited to take part in one or more rounds of a study. Attributes include:

- participant_id – unique identifier
- study_id – foreign key
- name, email – used for personalized invites and tracking

Participants are linked to the results through completed rounds.

5.3.5 Progress

Progress stores the sorting results for each participant in a given round. It includes:

- progress_id – identifier
- round_id, participant_id – composite foreign keys
- data – serialized or structured output of the sorting session
- progress – percentage of participant's completion

5.3.6 User

The User entity represents a system user who can create studies and manage participants. Attributes include:

- user_id – primary key
- organization_id – foreign key
- name, email, google_oauth_id – for login and user management

5.3.7 Permission

Each Permission entry defines a user's access rights to a particular study or entity. It includes:

- permission_id – identifier
- user_id – foreign key
- type – e.g., OWNER, EDITOR, VIEWER
- entity – refers to the type of resource (study, deck, etc.)

This supports fine-grained role-based access control via CASL.

5.3.8 Organization

Organizations group users by domain (e.g. university, institution). Attributes:

- organization_id – primary key
- domain, name – for institutional affiliation

This structure supports collaboration between multiple users from the same institution and enables domain-based onboarding.

5.4 Differences Between Design and Implementation

During the development phase, several changes were made to the original data model to better reflect practical needs and project constraints. These changes emerged as the application logic evolved and as new collaboration opportunities appeared.

One significant change was the introduction of independent card decks. Initially, card decks were directly linked to a study, but during implementation, it became clear that allowing users to create and manage card decks separately would provide greater flexibility. This way, users can prepare a set of cards without committing to a specific study immediately. As a result, the `study_id` foreign key in the `CardDeck` table was made nullable.

Due to time limitations, organization support was not implemented, even though it was included in the original data model. Although the Organization entity remains in the design, it was excluded from the final application logic to allow more focus on core study management features.

In addition, a new entity called UserPermission was introduced. This structure allows for more flexible and precise management of access rights across studies and related resources, and better supports role-based permissions using CASL.

After collaborating with another student whose application allows participants to complete Q-methodology studies, a new field called `presentation_config` was added to the `CardDeckVersion` entity [10]. This field stores display-related configuration (e.g. number of columns, slot distribution, instructional text), allowing the same card content to be presented differently across rounds or studies.

These modifications improved the system's flexibility, usability, and extensibility while maintaining alignment with the original design goals.

5.5 User Interface Design

The design of the user interface was inspired by modern web applications, with a focus on clarity, accessibility, and ease of use. The use case diagram introduced in the previous chapter provided a structural foundation for the key views and workflows required for the application. This chapter describes specific design decisions and features that were implemented in the system.

5.5.1 Overall Design Principles and Process

- **Material Design & Joy UI:** The interface is built using the Joy UI library, which is based on Material Design principles. It provides accessible and lightweight components suitable for admin interfaces.
- **User-Centered Iteration:** While some early mockups were created (e.g., login screen, dashboard), most of the interface was designed directly in code. This approach enabled faster iteration and tighter feedback loops. The design work was integrated with development, which aligned well with changing requirements and allowed quick adjustments based on informal user testing.
- **Design Goals:**
  - *Clarity and Simplicity* – Present key information without overwhelming users.
  - *Responsiveness* – Adjust layout and components for different screen sizes.

– *Separation of Roles and Contexts* – Provide clear task-based navigation to manage studies, rounds, cards and participants.
– *Scalability* – Ensure the layout and logic scale with more studies and participants.

5.5.2 Key Interface Features

Dashboard and Sidebar Navigation

The dashboard serves as the central overview page of the application. It displays aggregate statistics, such as the total number of studies, active rounds, and registered participants, in a clean and prominent layout. Below these statistics, users can find the latest rounds with information about their status, description, time remaining, and completion metrics. This allows researchers to quickly assess the status of ongoing studies without navigating to specific detail pages (Figure 5.3).

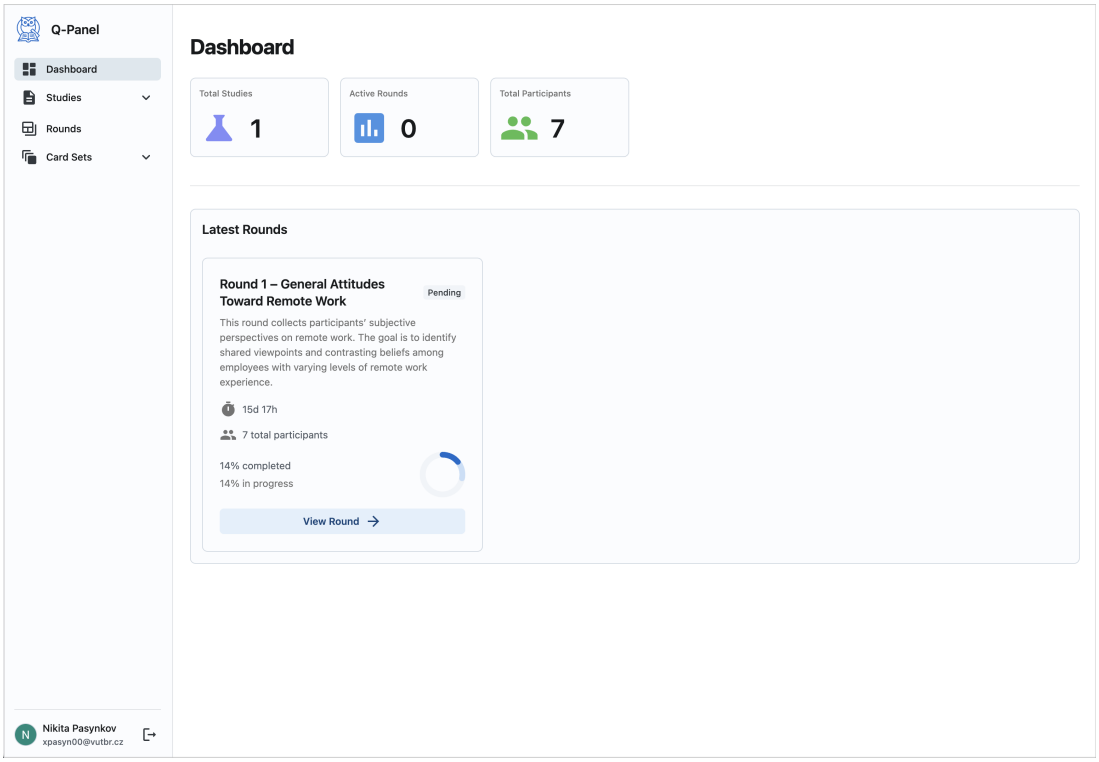

Figure 5.3: Current version of the dashboard displaying aggregate study metrics and latest round summary.

The sidebar provides a consistent and intuitive navigation experience. It includes links to all main sections of the admin panel: *Dashboard*, *Studies*, *Rounds*, and *Card Sets*. The sidebar also displays the current user's profile details at the bottom, enhancing user awareness and access to session controls such as logout. The structure of the sidebar ensures that users can easily navigate between views without getting lost in the interface hierarchy. This component contributes significantly to the overall usability and clarity of the application.

Study Creation in Steps

The process of creating a study is divided into clear steps: Study Details, Card Set, Round Details, and Participants (see Figure 5.4). This prevents users from becoming overwhelmed by too many inputs at once and makes the workflow easier to follow. It also allows the user to go back to previous steps without losing already completed data if needed.

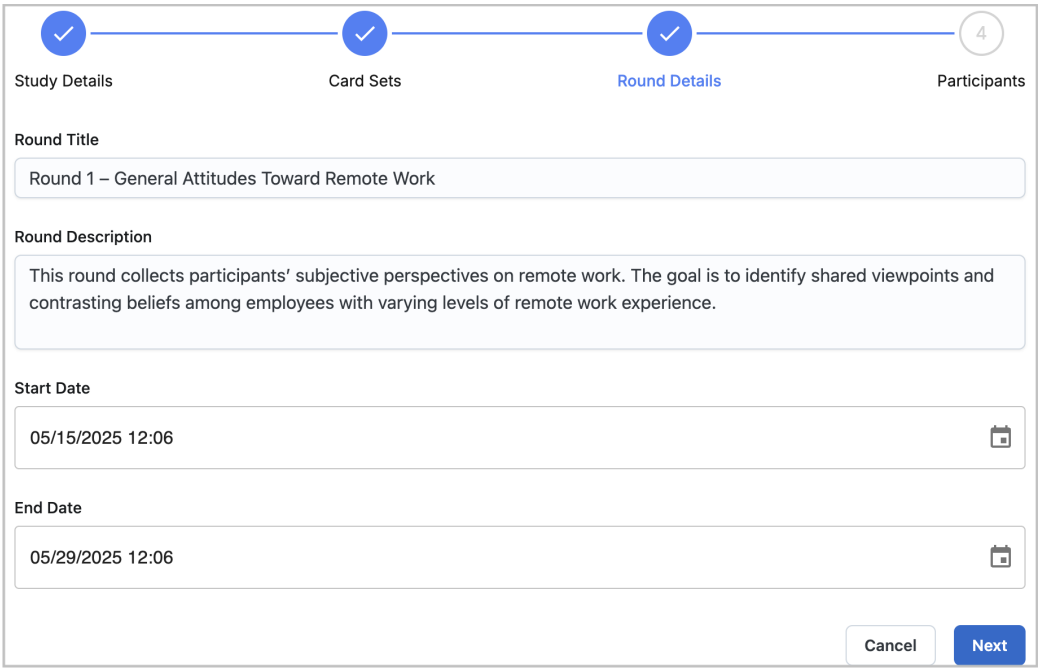

Figure 5.4: Step-by-step study creation process to reduce cognitive load.

Card Set Creation

Card sets are created using a form that supports naming the set, entering a sorting question, adding color-coded sections, and writing card statements (Figure 5.5). These sets can be reused across studies and edited later. It was changed several times during development as a result of collaboration with the sorting application developer, adding a new color tags field and optional colors for each section.

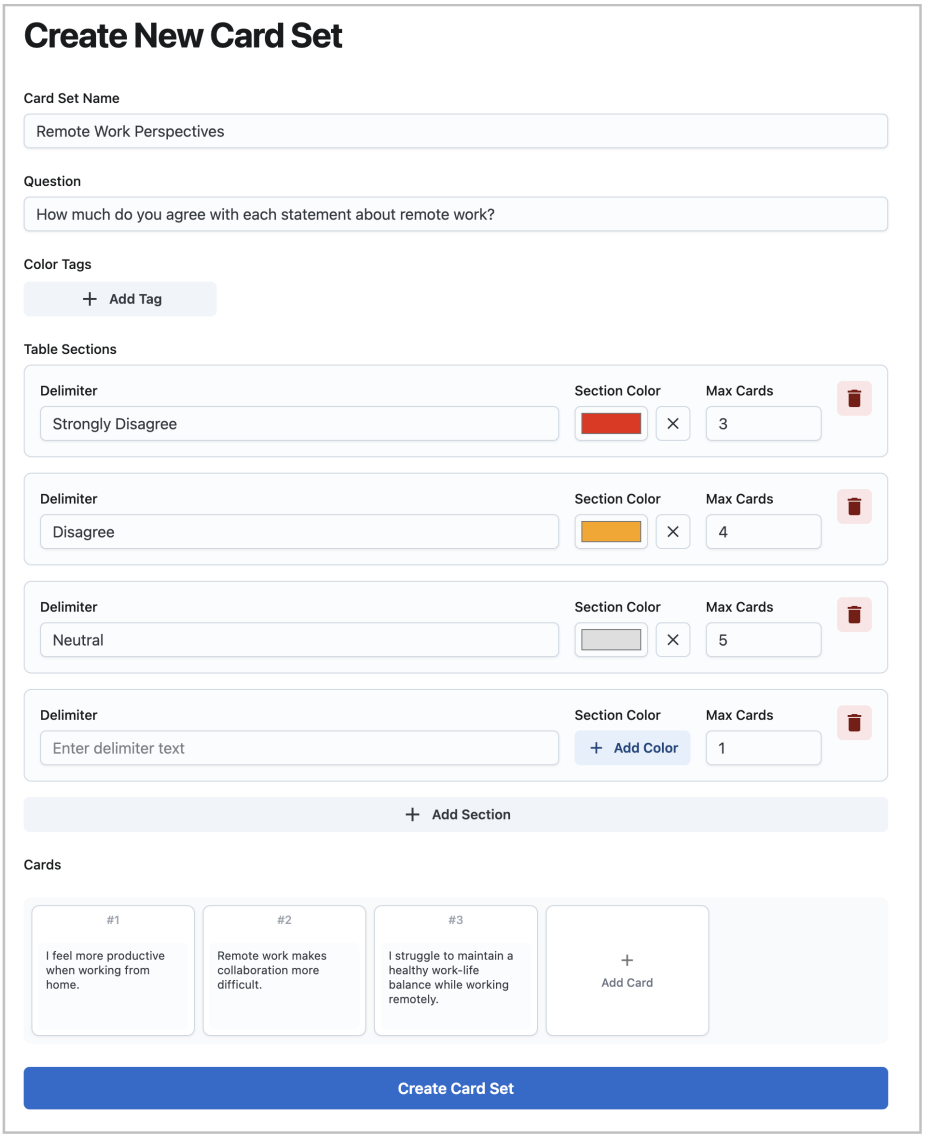

Figure 5.5: Form for creating a card set with sections and tags.

Versioning and Editing of Card Decks

The system uses versioning for card sets. When a user edits cards, the differences from the previous version are visually indicated, and individual changes can be reverted with a single button click. (Figure 5.6).

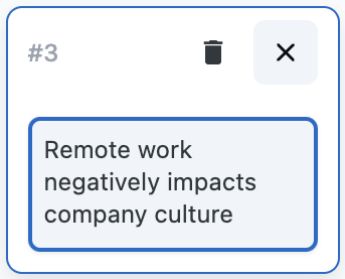

Figure 5.6: Editing a card with visible change and undo button.

Users can also switch between card deck versions. Previous versions are marked clearly, with edit options removed and a banner warning displayed (Figure 5.7).

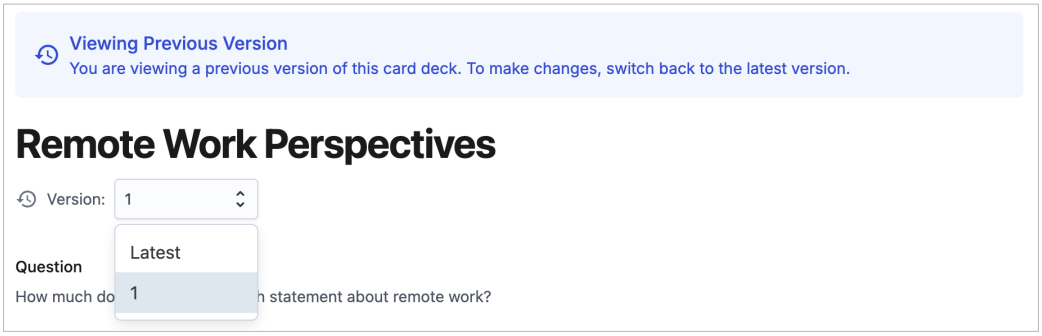

Figure 5.7: Previous version of a card deck with read-only view and banner.

5.5.3 Entity Tables

The application uses tables to display all of a user's studies, rounds, and card sets (Figure 5.8). These tables include sorting and filtering, improving usability for users with many entries.

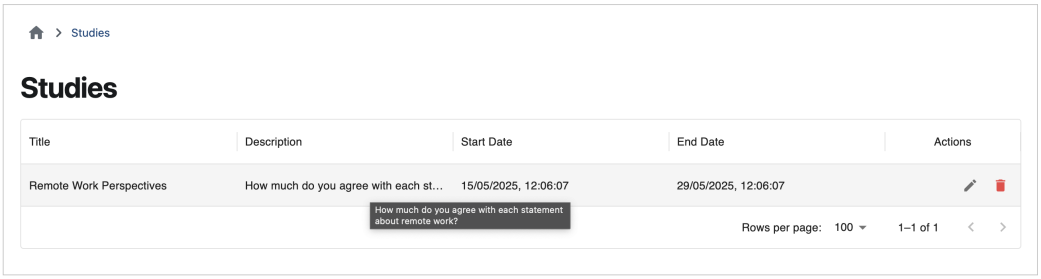

Figure 5.8: Study list with pagination, sorting, and tooltip descriptions.

5.5.4 Study Sharing Modal

A single modal is used for sharing studies with other users. From this view, a user can grant or revoke permissions, and manage access in one place (Figure 5.9).

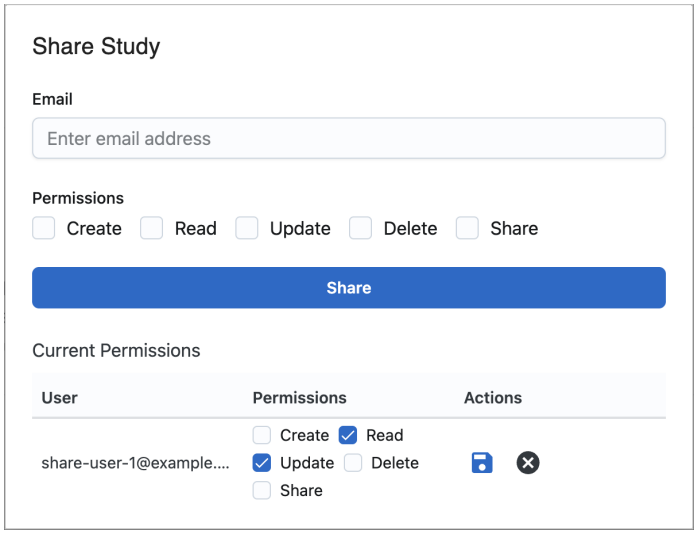

Figure 5.9: Study sharing modal with fine-grained permissions.

5.5.5 Participant Management Table

On the round detail page, administrators can manage study participants using a dedicated participant table interface. This table provides a clear overview of all participants invited to a specific round, including their email, progress percentage, last activity timestamp, and activity status. It also includes controls to exclude users from future rounds, generate or reset participation links, and re-invite inactive users.

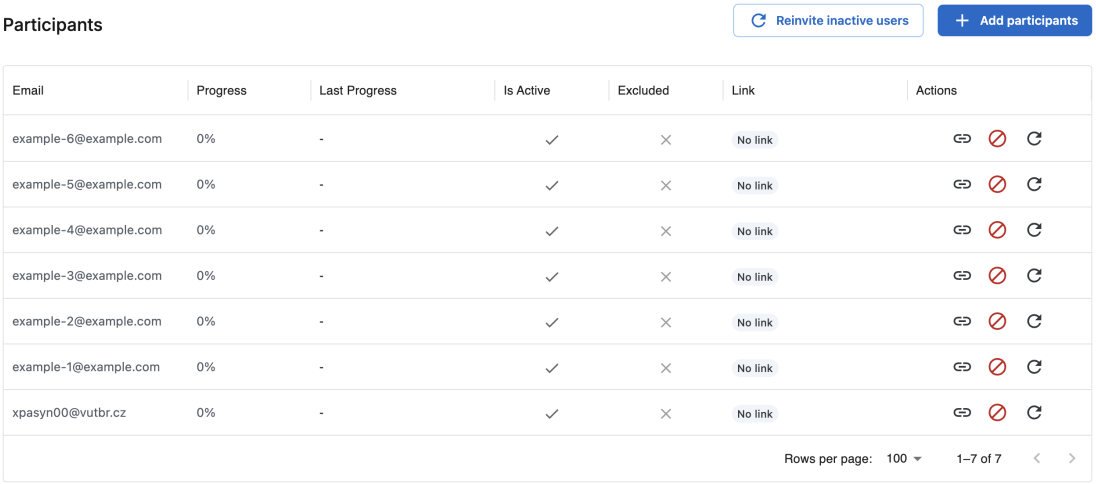

Figure 5.10: Participant management table on the round detail page.

Each row represents one participant, with icons for common actions: creating a unique link for a sorting session, excluding a participant from the current and future rounds, including a participant that was previously excluded, or resending their invitation email. The interface supports bulk re-invitation of inactive users through a button above the table. Status columns (such as *Is Active* and *Excluded*) use intuitive check and cross icons to improve readability.

This interface helps researchers monitor engagement in real time and take necessary action, such as excluding unresponsive users or resending links. It follows the permission-based logic of the application, disabling actions for users without the necessary rights.

Sign-In Page and Branding

The application features a clean and focused sign-in screen (Figure 5.11), supporting both email/password login and Google OAuth integration. Login and signup screens are kept minimal to reduce distractions. A distinctive owl logo is added on the left side and is reused on the dashboard header, reinforcing the application's branding and creating a friendly, trustworthy first impression for users.

This branding choice reflects research showing that consistent visual identity and symbolism can enhance user trust, recognition, and emotional connection to a product [9]. The owl, commonly associated with wisdom and learning, aligns naturally with the research-oriented context of Q-methodology studies.

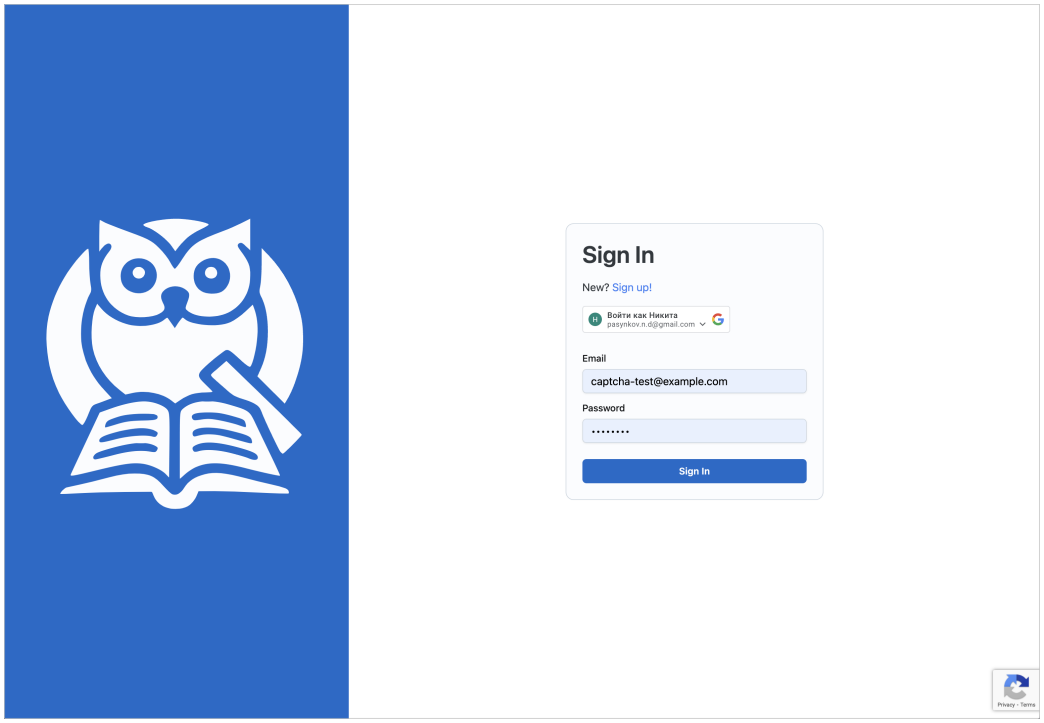

Figure 5.11: Sign-in screen with Google OAuth support, and owl branding.

5.5.6 Summary

The UI design of this application was heavily influenced by modern component libraries, user expectations for admin tools, and the practical workflows of researchers. By structuring complex tasks into simpler, step-based flows and applying reusable patterns like entity tables and modals, the application maintains clarity while offering powerful functionality. The use case diagram and data model guided the interface structure and provided a logical foundation for consistent UX.

Chapter 6

Implementation

6.1 Backend Implementation

The backend of the application is built with NestJS, offering a solid framework for managing the application's business logic, data handling, and API functions. This chapter covers the backend implementation, including its key components and design choices.

6.1.1 Modular Architecture

In accordance with NestJS's modular architecture, the infrastructure organizes related functionality into coherent modules. This approach enhances maintainability and allows a more efficient separation of concerns. The integration of specific feature modules is managed by the `app.module.ts` file, which serves as the application's entry point.

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'postgres',
      host: process.env.POSTGRES_HOST,
      port: 5432,
      username: process.env.POSTGRES_USER,
      password: process.env.POSTGRES_PASSWORD,
      database: process.env.POSTGRES_DB,
      schema: 'public',
      entities: [__dirname + '/**/*.entity{.ts,.js}'],
    }),
    ConfigModule.forRoot({ isGlobal: true }),
    AuthModule,
    CaslModule,
    UsersModule,
    StudiesModule,
    RoundsModule,
    CardDecksModule,
    // Other modules...
  ],
  controllers: [AppController],
})
```

```
export class AppModule {
}
```

Each module incorporates a specific domain area, such as user management, card decks, rounds, or studies, with its own controllers, services, and entities. This structure aligns with domain-driven design principles, making the codebase simpler to comprehend and maintain [7].

6.1.2 Database

PostgreSQL serves as the primary data store for the application, and TypeORM implements an object-relational mapping layer. The database schema is initialized using migration scripts, which ensures a consistent database structure across a variety of environments. The fundamental data model is established by the migration scripts that define entities such as users, studies, rounds, card decks, and participants. This relational model captures the relationships between the various entities involved in the Q-methodology research process.

```
CREATE TABLE studies
(
    id uuid DEFAULT gen_random_uuid() PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    description VARCHAR(255) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    user_id uuid NOT NULL,
    CONSTRAINT fk_user_id FOREIGN KEY (user_id) REFERENCES users (id)
               ON UPDATE CASCADE ON DELETE CASCADE
)
```

Listing 6.1: Example from 007-Study.sql

6.1.3 Authentication and Authorization

The authentication system utilizes a JWT-based authentication approach, which supports both conventional email/password login and Google OAuth integration. Bcrypt hashing is used to ensure the security of user credentials [1], and refresh tokens enable session management.

CASL, which provides granular access control based on user roles and permissions, is used in authorization implementation. Flexible access policies are made possible because of this implementation, which is essential in cooperative research environments where users may have varying levels of access to studies and their associated entities.

6.1.4 Studies and Rounds Administration

The `StudiesModule` and `RoundsModule`, which together form the application's primary functionality, are responsible for the creation, administration, and execution of Q-methodology studies.

- **Studies**: Manages round relationships, study metadata, and permissions.
- **Rounds**: Is responsible for the management of individual research rounds, which include their lifecycle phases (pending, active, and completed).

The rounds system includes a scheduler component that provides automated status updates:

```
@Injectable()
export class RoundsScheduler {
  @Cron(CronExpression.EVERY_DAY_AT_MIDNIGHT)
  async handleRoundStatusUpdates() {
    try {
      const { updatedActiveRounds, updatedCompletedRounds } =
        await this.roundsService.updatePendingRounds();

      // Logging and error handling
    } catch (error) {
      this.logger.error('Error updating round statuses:', error?.stack);
    }
  }
}
```

By automatically switching rounds between states according to pre-configured start and end dates, this scheduler saves researchers administrative work.

6.1.5 Card Deck and Card Management

The `CardDecksModule`, `CardsModule`, and `CardDeckVersionsModule` card management components provide the ability to create, version, and manage Card Decks and statement cards that are used in Q-sorts.

- **CardDecks**: Manages groups of connected cards, including all previous versions.
- **Cards**: Oversees the metadata and content of each statement card.
- **CardDeckVersions**: Allows iterative development by integrating versioning for card decks.

This versioning feature is particularly beneficial in Q-methodology research, as it enables researchers to modify their statement sets in response to feedback or experimental studies while maintaining the integrity of ongoing studies.

6.1.6 Participant Management

Participant-related services (`ParticipantsModule`, `StudyParticipantsModule`, `RoundParticipantsModule`) handle the invitation, tracking, and management of research subjects:

- **Participants**: Manages participant records and metadata.
- **RoundParticipants**: Monitors participant engagement in specific rounds.

The RoundParticipantsService includes functionality for tracking completion rates and progress, providing researchers with real-time insights into data collection status.

6.1.7 Round Link and Progress Tracking

Two additional modules were implemented to help with communication with the external participant-facing application: `Progress` and `RoundLink`. These modules are necessary for the purpose of associating individual participants with their activity throughout the Q-sort process.

RoundLink Module

The RoundLink module is responsible for the generation, retrieval, and validation of distinct links that establish a connection between a participant and a particular round. Each link is associated with a combination of a `participant_id` and `round_id`.

This entity serves mainly for helping with interaction between the administrator platform and the external application for the participants used to conduct the Q-sort, as well as to send personalized invitations to participants [10].

The external application sends a request to the `round-links.controller.ts` when a participant accesses the provided link, which in turn retrieves the associated round configuration. An additional API token was generated to prevent users from accessing data that is intended only to be accessible for the sorting application.

The link serves as the entry point for launching a sorting session. Existing progress can be obtained from the backend if the participant has previously interacted with the round. If not, the latest version of the card deck associated with the round is used to initiate a new session.

Progress Module

The Progress entity is intended to monitor the ongoing activity of a participant within a particular round. Included in this are three primary characteristics:

- **data**: a JSON field that contains arbitrary metadata regarding the Q-sort operation
- **progress**: a numerical percentage that indicates the percentage of statements that have been sorted
- **completed**: a boolean value that indicates whether the participant has finished their sorting session(s)

Endpoints for viewing and writing progress information are managed by the `progress.controller.ts` and `progress.service.ts`. The backend is able to store real-time data regarding participant engagement as a result of the external application calling a POST endpoint after each session.

The administrator panel relies on this progress data to offer researchers an overview of the activity of individual participants.

In addition, a scheduler is implemented to identify inactive participants by analyzing the absence or delay of their progress updates. This helps researchers exclude participants from taking part in the following rounds based on their activity.

6.1.8 File Management

The `FileModule` is responsible for the storage and upload of media, including image and document attachments. Although the application currently only uses this module for storing

Google account profile picture, it was implemented to support future application improvements, such as attaching images to a statement card or user profile settings.

6.1.9 API Endpoints

The backend provides RESTful API endpoints that comply with consistent resource management patterns:

- **GET endpoints**: Retrieve entities and their relationships
- **POST endpoints**: Create new resources
- **PATCH/PUT endpoints**: Update existing resources
- **DELETE endpoints**: Remove resources when appropriate

These endpoints implement proper validation, error handling, and authorization checks, ensuring data integrity and security.

To simplify development and improve usability for external developers, API documentation is automatically generated using the Swagger module for NestJS. The documentation is available at `https://backend.qpanel.settler.tech/docs`.

This interface includes:

- Descriptions for all available endpoints
- Input and output schemas
- Authentication requirements and example payloads

Swagger decorators such as `@ApiBody` or `@ApiResponse` are used to annotate endpoints and ensure the generated documentation remains synchronized with the codebase:

```
@Post(:roundId)
@HttpCode(HttpStatus.CREATED)
@ApiBody({
    type: () => CreateRoundParticipantsDto,
    required: true,
})
@ApiCreatedResponse({
    type: () => RoundParticipant,
})
```

This integration improves collaboration and testing workflows by offering a self-updating, interactive reference for developers and contributors.

6.1.10 Docker Integration

Docker[1] is a platform for developing, shipping, and running applications in lightweight, isolated containers that ensure consistency across different environments.

[1] `https://www.docker.com/`

Backend Containerization

The backend is built using NestJS and containerized with the following Dockerfile:

```
FROM node:20-alpine as backend

WORKDIR /app

# Install build tools and dependencies
RUN apk --no-cache add --virtual builds-deps build-base python3

# Copy package and config files
COPY package.json .
COPY yarn.lock .
COPY nest-cli.json .
COPY tsconfig.json .
COPY tsconfig.build.json .

# Install dependencies
RUN yarn install

# Copy all source code and build it
COPY . .
RUN yarn build

# Expose port and start the app in production mode
EXPOSE 3000
CMD ["sh", "-c", "yarn start:prod"]
```

This configuration ensures that all necessary packages are installed in an Alpine-based environment and that the compiled application is started in production mode using Yarn[2].

Database Containerization

Although PostgreSQL itself runs as a standard image using Docker Compose, a separate Dockerfile is provided for running database-related operations such as migrations using TypeORM:

```
FROM node:20-alpine

WORKDIR /app

RUN apk --no-cache add --virtual builds-deps build-base python3

COPY package.json .
COPY yarn.lock .
COPY nest-cli.json .
COPY tsconfig.json .
COPY tsconfig.build.json .
```

[2] https://yarnpkg.com/

```
RUN npm i -g typeorm

RUN yarn install
```

This Dockerfile allows running database schema migrations in an isolated and reproducible environment.

6.1.11 Permission System

The backend implementation establishes a flexible and robust foundation for the entire application by integrating modern architectural patterns with domain-specific considerations for Q-methodology research.

6.1.12 Interceptor-Based Permission System

The implementation of the permission system is one of its most innovative features, as it implements NestJS interceptors to manage complicated authorization requirements. This method was developed to resolve specific challenges presented by application design: it must be possible for user to grant access to all study-related entities (rounds, card decks, participants) from one simple action, without configuring permissions separately for each entity connected to the study.

6.1.13 Definition of Interceptor

Before introducing the system, it is important to understand what interceptors are in the context of NestJS. Interceptors are a special type of class that are positioned between the request and the controller method, allowing developers to execute custom logic before or after the request is handled. Common use cases include logging, transforming responses, caching, and authorization [3].

In this project, interceptors are used to enforce permissions in a reusable and declarative way. Unlike middleware or guards, interceptors can modify both the request and response, making them suitable for more complex access control logic that depends on runtime entity data. The system uses two primary interceptors that collaborate to establish a sophisticated permission-checking workflow:

6.1.14 Implementation of AttachEntityInterceptor

AttachEntityInterceptor: It is responsible for locating the appropriate entity and attaching it to the request.

This interceptor:

- Based on the request path, determines the entity type that is being accessed.
- Searches through the database to find the particular entity instance.
- Attaches the entity to the request object for future permission verification.

The interceptor is crucial for the preservation of permission inheritance across related entities, as it manages both direct entity access and nested entity relationships.

CheckAbilitiesInterceptor: Determines whether the user has the necessary permissions for specific actions related to those entities. This method provides several benefits in comparison to authorization that is based on traditional middleware:

This interceptor:

- Makes use of CASL to generate ability rules for the current user.
- Determines whether the user has the necessary permissions to perform the specified actions.
- Filters the response data to include only the entities that the user has permission to access.
- Throws an exception when authorization is denied.

6.1.15 Usage in Controllers

All of these interceptors are being used alongside one another to enforce permissions in controller endpoints:

```
@Delete('/:id')
@UseGuards(JwtAuthGuard)
@UseInterceptors(
  AttachEntity(StudiesService, (request) => ({
    [E_STUDY_ENTITY_KEYS.ID]: request.params.id,
  })),
  CheckAbilities({ action: E_ACTION.DELETE }),
)
async delete(@Param('id') id: string): Promise<void> {
  // Deletion logic...
}
```

6.1.16 The Advantages of the Interceptor Approach

This permission system, which depends on interceptors, provides many benefits:

- **Hierarchical Permissions**: When a user is granted access to a study, they automatically gain access to all related entities, simplifying permission management.
- **Separation of Concerns**: The permission logic is separated from the business logic, keeping the controllers clean.
- **Reusability**: The interceptors can be reused across different controllers and endpoints, ensuring consistent permission enforcement.
- **Fine-grained Control**: Permissions can be verified at various levels of detail, ranging from broad entity classes to specific entity instances.
- **Response Filtering**: The system automatically filters responses to include only entities the user has permission to access, preventing information leakage.

6.1.17 Challenges during Implementation

The development of this permission system introduced some challenges:

- **Complex Entity Relationships**: Mapping permissions across nested entity relationships required careful design and extensive testing.
- **Performance Requirements**: The system was required to efficiently verify permissions without generating an excessive number of database queries.
- **Edge Cases**: The thorough testing required to address a variety of permission scenarios, including users with varying roles within the same study.

The interceptor-based permission system is a major innovation in the application's architecture, as it enables flexible, hierarchical permission administration that aligns perfectly with the collaborative nature of Q-methodology research.

6.2 Frontend Implementation

This chapter provides a comprehensive overview of the frontend implementation in the Q-methodology application.The key components and patterns will be described using the Round entity as an example.

6.2.1 API Services

API services in the frontend application follow a structured approach that encapsulates communication with the backend. Each entity has its own dedicated service class that defines endpoints for CRUD operations. The following example shows the `RoundService` implementation.

```
export default class RoundService extends BaseService {
  protected static readonly endpoint = '/rounds';

  static async create(
    studyId: TRound[E_ROUND_ENTITY_KEYS.STUDY_ID],
    data: TRoundCreateData
  ): Promise<TRound> {
    return await Api.instance.post<TRoundCreateData, TRound>(
      `${this.endpoint}/${studyId}`,
      data
    );
  }
}
```

The service implements standard methods for fetching, creating, updating, and deleting rounds. It uses a centralized `Api.instance` that manages HTTP requests with Axios. This design provides several advantages:

1. **Centralized Error Handling**: All API requests go through a single API instance
2. **Type Safety**: Methods are properly typed with TypeScript, ensuring data consistency

3. **Reusability**: The service can be used across different components

4. **Abstraction**: Components do not need to know the details of API implementation.

Similarly, other entities follow the same pattern, providing methods for entity-specific operations.

6.2.2 Data Fetching Hooks

Tanstack React Query is used for data fetching, providing built-in caching, loading states, and error handling. Custom hooks are created for each entity to encapsulate the data fetching logic. For example, `useRound` hook:

```
export const useRound = (
  id: TRound[E_ROUND_ENTITY_KEYS.ID],
  options?: Omit<
    UseQueryOptions<TRound, TApiError, TRound, Array<string>>,
    'initialData' | 'queryFn' | 'queryKey'
  > & { initialData?(): undefined }
) =>
  useQuery({
    queryKey: [`${E_ENDPOINTS.ROUNDS}#getOne`, id],
    queryFn: async (): Promise<TRound> => RoundService.getOne(id),
    ...options,
  });
```

This hook uses `useQuery` from React Query, with a `queryKey` for caching and a `queryFn` as a function that fetches data. The `queryKey` includes the entity type and ID, enabling React Query to cache the results. Using React Query together with Next.js allows the web application to perform data fetching and updates without requiring full page reloads, resulting in a smoother and more responsive user experience.

Additional hooks are created for more specific data fetching needs, allowing components to easily access the data they need without duplicating fetching logic.

6.2.3 Mutation Hooks

The React Query mutation hooks handle data mutations (create, update, and delete). The following hook provides a set of mutations for managing rounds:

```
export const useRoundMutations = ({
  refetch,
}: TUseRoundMutationsParams): TUseRoundMutations => {
  const createMutation = useMutation<
    TRound,
    TApiError,
    TRoundCreateMutationVariables
  >({
    mutationFn: async ({
      [E_ROUND_ENTITY_KEYS.STUDY_ID]: studyId,
      data,
    }: TRoundCreateMutationVariables) => RoundService.create(studyId, data),
```

```
    onSuccess: async () => {
      await refetch();
      toast.success('Round created successfully');
    },
    onError: async () => {
      await refetch();
      toast.error('Failed to create round');
    },
  });

  // Similar implementation for update and delete mutations

  return {
    createMutation,
    updateMutation,
    deleteMutation,
  };
};
```

Each mutation (`createMutation`, `updateMutation`, `deleteMutation`) uses the `useMutation` hook, with:

- A `mutationFn` to perform the API request using the appropriate service method
- `onSuccess` and `onError` callbacks to refetch data and provide user feedback via toast notifications

This approach provides several benefits:

1. **Consistent Feedback**: Users receive consistent feedback via toast notifications
2. **Automatic Data Refreshing**: Successful mutations trigger a refetch of the data
3. **Type Safety**: TypeScript ensures data consistency throughout the mutation process
4. **Reusability**: Mutations can be used in different components

For more complex entities, mutation hooks may include additional mutations to handle specific use cases.

6.2.4 Pages Implementation

The application follows the Next.js `app` directory structure, allowing server-side rendering, static site generation, and client-side rendering. The following example shows the implementation of a Rounds page, which is similar to any other general entity page:

Rounds Page `/app/(main)/rounds/page.tsx`

The page displays all rounds in a table format, allowing users to view, edit, and delete them:

```
const RoundsPage = () => {
  // Entity hooks for fetching data and mutations are declared here

  return (
    <Container>
      <Typography level="h1">Rounds</Typography>
      <GenericTable
        columns={columns}
        primaryKey={E_ROUND_ENTITY_KEYS.ID}
        caption="Card Sets"
        queryFunction={useRounds}
        mutationsFunction={useRoundMutations}
        route={E_ROUTES.ROUNDS}
        actions={['delete']}
      />
    </Container>
  );
};
```

This page uses:

1. The `useRounds` hook to fetch all rounds
2. Hooks to fetch additional data (participant counts) for each round
3. The `GenericTable` component to display the rounds in a table format
4. The `useRoundMutations` hook for managing round mutations (delete)

The component is a reusable table that handles common operations such as pagination, sorting, and actions (edit, delete). It takes the query function and the mutations function as props, allowing it to work with any entity.

Round Detail Page (`/app/(main)/rounds/[roundId]/page.tsx`)

Dynamic routing is used to create resource-specific pages. The RoundPage component is placed inside the `[roundId]` folder to support dynamic routes in Next.js, where square brackets define a variable segment of the URL. As a result, any path such as /rounds/123 or /rounds/abc is routed to this page, and the corresponding value is passed into the component through the `params` object (e.g., { roundId: „123“ }).

This approach allows the application to display different content based on the URL, without the need for additional parsing or query parameters. It improves maintainability by aligning the folder structure with the routing structure and also ensures clean URLs. The `params.roundId` value is then used to fetch the appropriate round data within the component logic.

The round detail page displays detailed information about a specific round and allows users to edit it:

```
const RoundPage = ({ params }: TRoundPageParams) => {
  // Editing is handled on the same page
  const [isEditing, setIsEditing] = useState(false);
```

```
  // Data is fetched base on the round id in the URL
  const { data, isLoading, refetch } = useRound(params.roundId);

  // Permissions for actions such as editing are fetched
  // to disable actions for users without necessary permissions
  const { hasUpdatePermission } = useStudyPermissions(studyId || '');

  const { updateMutation } = useRoundMutations({
    refetch,
  });


  // Handle form submission
  const handleSave = () => {
    updateMutation.mutate({
      // Data for updating an entity
    });
  };

  return (
    // The layout and additional components
  )
};
```

This page presents several key patterns:

1. **Data Fetching**: The `useRound` hook fetches the round data
2. **Permission Checking**: The `useStudyPermissions` hook checks if the user has permission to update the round
3. **Mutations**: The `updateMutation` hook, mentioned in Section 6.2.3 handles updating the round
4. **Integrated interaction**: Editing and viewing are handled within the same page, depending on user permissions.

6.2.5 Additional Key Elements

The frontend implementation showcases several key patterns and practices:

1. **Type Safety**: TypeScript is used throughout the application to ensure type safety and provide better development experience.
2. **Component-Based Architecture**: The application is built using reusable components.
3. **Separation of Concerns**: Data fetching, mutations, and UI rendering are separated.
4. **Consistent Error Handling**: Errors are handled consistently across the application.

5. **Responsive Design**: The application works well on different screen sizes.

6. **Performance Optimization**: React Query's caching and state management improve application's performance.

6.3 Email Service Integration and Domain Verification

A critical part of the Q-methodology application is the email service, which enables researchers to invite participants to studies and rounds. The implementation includes both development and production email configurations, with special considerations for domain verification and anti-spam protection. Implementation of this feature required working with both frontend and backend, as well as with Google Cloud console, which is why it is described in a separate section.

6.3.1 Email Service Architecture

The email functionality is encapsulated in the `EmailModule`, which provides services for sending templated emails to study participants. The service uses environment detection to switch between development and production configurations:

```
export class EmailService implements OnModuleInit {
  private isProduction: boolean;

  async onModuleInit() {
    this.isProduction =
      this.configService.get('APP_URL') === 'https://backend.qpanel.settler.tech';

    if (this.isProduction) {
      // Use Mailtrap for production
    } else {
      // Use local SMTP for development
    }
  }
}
```

The service provides specialized methods for sending bulk invitations to study participants with a pre-defined HTML email template:

```
async sendBulkRoundInvitations(
  invitations: Array<{
    email: string;
    roundTitle: string;
    studyTitle: string;
    link: string;
  }>
): Promise<Array<any>> {
  // Method implementation
}
```

This method processes invitations in batches to optimize performance and reliability, handling potential errors to ensure the invitation process continues even if individual emails fail.

6.3.2 Domain Verification and Anti-Spam Measures

Integrating with Mailtrap[3] as the production email service provider required domain verification to ensure legitimate email sending. As part of this process, the application had to include anti-spam measures to maintain a sender reputation and prevent misuse.

A key requirement from Mailtrap was the implementation of CAPTCHA protection on public-facing forms to prevent automated bot submissions. To address this requirement, Google reCAPTCHA v2 was implemented for all public authentication forms (sign-up and sign-in). The implementation consists of:

- **Backend Guard:** A `RecaptchaGuard` that validates tokens from the frontend

```
@Injectable()
export class RecaptchaGuard implements CanActivate {
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const { recaptchaToken } = request.body;

    // Validate token with Google reCAPTCHA API
    // ...
  }
}
```

- **Frontend Integration:** Adding reCAPTCHA components to authentication forms and including tokens in requests
- **Guard Application:** Applying the guard to relevant authentication endpoints

This implementation satisfied the email provider's requirements for anti-spam protection, allowing the domain verification process to proceed. The verification process then involved:

1. Creating DNS records to prove domain ownership
2. Configuring SPF, DKIM, and DMARC records to authenticate email sending
3. Completing the verification process with Mailtrap

6.3.3 Technical Challenges and Solutions

Implementing the email system presented several technical challenges:

1. **Environment-Specific Configuration:** The system needed different email configurations for development and production. This was addressed through environment detection and conditional service configuration.

[3] `https://mailtrap.io/`

2. **Bulk Sending Optimization:** Sending invitations to multiple participants required optimization to avoid rate limits. The implementation uses batching to process invitations in smaller groups.

3. **Google reCAPTCHA v2 Integration:** Implementing Google reCAPTCHA presented challenges due to slightly non-linear documentation and API nuances [5]. The implementation required:

   - Understanding the distinction between frontend and backend verification flows
   - Correctly formatting the verification request as form-encoded data rather than JSON
   - Handling the various error responses that may be returned from the API
   - Creating a proper error handling mechanism that provided meaningful feedback

   The solution involved a custom NestJS guard that encapsulates the verification logic:

```
@Injectable()
export class RecaptchaGuard implements CanActivate {
  async canActivate(context: ExecutionContext): Promise<boolean> {
    const request = context.switchToHttp().getRequest();
    const { recaptchaToken } = request.body;

    if (!recaptchaToken) {
      throw new ForbiddenException('reCAPTCHA token is required');
    }

    try {
      const secretKey =
        this.configService.get<string>('GOOGLE_RECAPTCHA_SECRET_KEY');
      const formData = new URLSearchParams({
        secret: secretKey,
        response: recaptchaToken,
      }).toString();

      // Make verification request to Google's API
      // ...

      return true;
    } catch (error) {
      // Throw an error with appropriate message
    }
  }
}
```

4. **Error Handling:** Email-sending failures needed handling to ensure success. The service implements `Promise.all` with error catching to handle individual email failures.

The integration of email services demonstrates the importance of considering both functional requirements and security/compliance requirements when implementing production

systems. The inclusion of reCAPTCHA v2 protection not only satisfied the email provider's requirements but also enhanced the overall security of the application by preventing automated account creation.

Chapter 7

Testing and Evaluation

One of the goals of the project was to iteratively improve the user interface and overall usability. Although the original plan included feedback from people who were experienced with Q-methodology or research workflows, I was unable to reach users who worked directly in academic research. Instead, I asked several people with general technical knowledge to provide feedback after testing application's features.

7.1 Informal Usability Feedback Process

Feedback was collected in an informal setting. Participants were asked to perform basic tasks such as the following:

- Create a new study
- Edit a card set
- Start a round
- View participant results

Although these users were not familiar with the Q-methodology specifically, they were able to evaluate how intuitive and understandable the interface was, which helped identify areas of friction or confusion. Their comments, along with my own observations during development, led to a number of design improvements.

7.2 Key Changes Based on Feedback

- **Study creation restructured into steps**

  Initially, the entire process of creating a study was placed on one page, making the form too complex and difficult to navigate. Based on usability concerns, the process was split into four clear steps: study details, card set selection, round configuration, and participant management. This change simplified the experience and reduced user error (see 5.3).

- **Simplified layout and visual adjustments**

The original layout included both a sidebar and a top navigation bar. Feedback suggested that this made the application look cluttered and outdated. The top navigation bar was removed and the layout was redesigned with only a sidebar and a cleaner color scheme. In addition, font sizes were adjusted to improve readability.

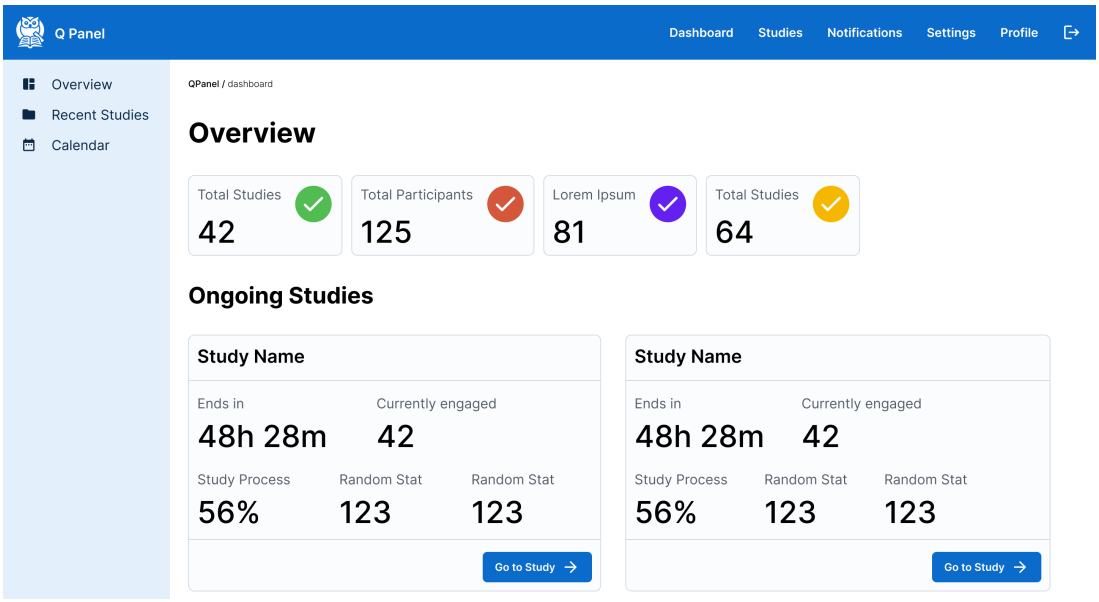

Figure 7.1: Initial version of the dashboard. This version used a different layout and color scheme. See Figure 5.3 for the current version.

- **Responsiveness**

  Some users noticed layout issues on smaller screens. Improvements were made to the responsiveness of most pages, although more work is still needed. Due to time constraints and the system's primary use on desktop, full mobile support is planned for future development.

- **Removed animations**

  Subtle UI animations were originally added for buttons and inputs, but they did not align well with the feel of an administrator panel interface. After review, they were removed to keep the interface clean and free of distractions.

- **Component tweaks and permission-based UX**

  Minor changes were made based on feedback about button positions, component sizes, and layout consistency. Additionally, logic was added to disable buttons or actions for users without the correct permissions, making the app feel more user-friendly.

7.3 Conclusion

Although no formal usability testing was conducted, I relied on informal feedback and my own experience using the application to guide improvements. This iterative approach led to a number of meaningful changes that improved the clarity, consistency, and overall usability of the system. Future work should include structured testing with real re-

searchers and usability-focused interviews to validate design decisions in the context of real Q-methodology workflows.

7.4 API Testing

In the early stages of development, the backend API was tested using Insomnia, allowing for quick iteration and verification of basic endpoint functionality.

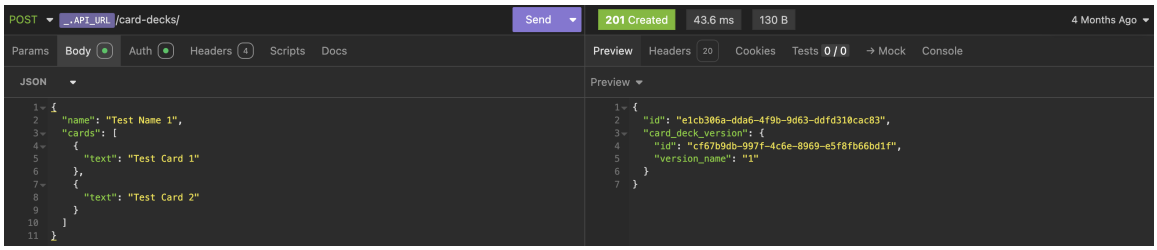

Figure 7.2: Creating a new card deck with Insomnia. A simple POST request was used to test the endpoint and verify that the structure and versioning logic were working as expected.

As more complex features such as permission handling were introduced, unit tests were implemented using Jest. However, since the structure of many backend modules and entities changed frequently during development, maintaining Jest tests became too time-consuming.

Instead, Postman's "Flows" feature was used for testing, which allowed to chain requests and reuse data between them. This approach appeared to be more flexible and suited for testing realistic scenarios, such as end-to-end workflows involving authentication, study creation, permission sharing, and participant progress submission.

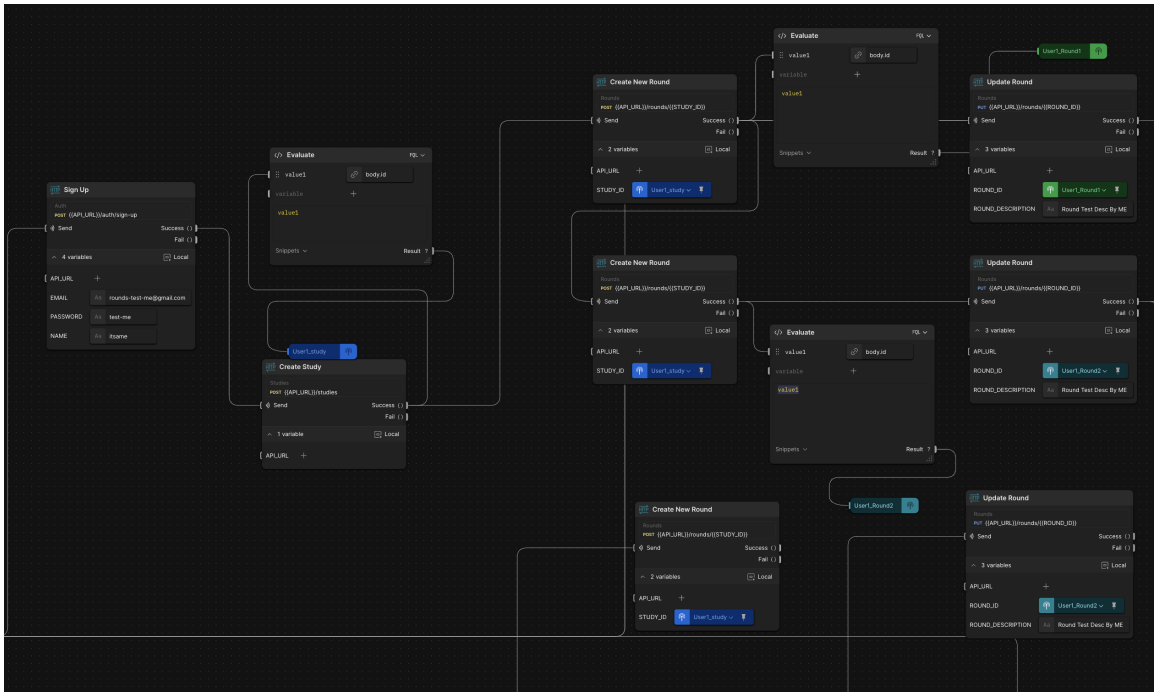

Figure 7.3: Example of a Postman Flow for creating and updating rounds. This is only a minor part of a larger flow that tests the permission system.

Chapter 8

Conclusion and future improvements

The final result of this project can be considered successful. The key functionalities that were planned at the beginning of development were implemented, and the application is fully usable as an admin interface for managing Q-methodology studies. The developed application is now fully operational and available at `https://qpanel.settler.tech/`. The system supports study and round management, card set creation and versioning, participant tracking, email support, and permission-based access control. Although certain parts could be improved upon, the core goals of the project were met.

During implementation, the scope was iteratively adjusted to reflect available time and resources. Several advanced or optional features discussed with the thesis supervisor were postponed. These include, for example, a monetization feature for card sets - allowing researchers to create high-quality card sets and sell them to others - which would require significant changes to the way card sets are shared and additional safety measures. Other ideas such as email address validation based on Google organization domains, suggesting participants based on the previous user's session, or more advanced email features (e.g. custom message templates or scheduled email dispatching) remain on the list of future improvements.

The project also involved close collaboration with another student, who developed the participant-facing interface where Q-sorts are performed [10]. A key part of this cooperation was the implementation of round links - unique URLs generated by my system that allow participants to access the sorting interface built by the other student. We coordinated features and API requirements to ensure smooth data exchange between both systems. When a participant completes a sorting session, the results are sent back to my application, allowing researchers to track user progress in real time. Thanks to this collaboration, both applications can now be deployed together and used to conduct Q-methodological studies.

In terms of technical enhancements, one of the main areas for further work is responsiveness and mobile optimization. Although the application works on smaller screens to some extent, certain views still require visual adjustments to ensure a seamless experience across all device sizes.

Another area worth expanding is the email functionality. Currently, the system supports sending invitations and notifications, but in the future, it could allow more flexible email customization, scheduling, and improved logging of sent messages.

Regarding the analysis of completed research, the addition of analytical tools for researchers was also considered. This could include visual dashboards and statistical summaries at the round level, allowing users to gain deeper insights into participant activity and Q-sort distributions through charts and graphs.

Finally, the application would benefit from broader test coverage, particularly on the backend. Due to the fast pace of development and frequent changes in the application logic, it was not feasible to maintain a full suite of Jest tests throughout the project. Now that the core functionality has stabilized, writing a more complete set of unit and integration tests would help support future updates and improve maintainability.

Although there are a variety of possible improvements, the current state of the application meets the initial expectations and succeeds in building a solid foundation for planned updates. The interface is intuitive and clean, and the application is ready for practical use in real research workflows.

Bibliography

[1] Arias, D. *Hashing in Action: Understanding bcrypt* online. 2021. Available at: https://auth0.com/blog/hashing-in-action-understanding-bcrypt/.

[2] Brown, S. R. *Political Subjectivity: Applications of Q Methodology in Political Science*. Yale University Press, 1980.

[3] Contributors, N. *Interceptors | NestJS - A progressive Node.js framework*. 2024. Available at: https://docs.nestjs.com/interceptors.

[4] Developers, G. *OAuth 2.0 for Web Server Applications* online. Available at: https://developers.google.com/identity/protocols/oauth2.

[5] Developers, G. *ReCAPTCHA v2* online. Available at: https://developers.google.com/recaptcha/docs/display.

[6] Documentation, T. *What is the Data Mapper pattern?* online. 2023. Available at: https://typeorm.io/active-record-data-mapper#what-is-the-data-mapper-pattern.

[7] Evans, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional, 2003. ISBN 978-0321125217.

[8] JWT.io. *Introduction to JSON Web Tokens* online. 2025. Available at: https://jwt.io/introduction.

[9] Keller, K. L. and Richey, K. The importance of corporate brand personality traits to a successful 21st century business. *Journal of Brand Management*, 2006, vol. 14, no. 1.

[10] Pavelka, V. *Research and Development of UI/UX for Q Sorting* online. 2025. Available at: https://www.vut.cz/en/students/final-thesis/detail/163013. Bachelor's Thesis. Supervisor: Adam Herout.

[11] Stephenson, W. *The Study of Behavior: Q-Technique and Its Methodology*. University of Chicago Press, 1953.

[12] Watts, S. and Stenner, P. Doing Q Methodological Research: Theory, Method and Interpretation. *Qualitative Research in Psychology*, 2005, vol. 2, no. 1.

Appendix A

Poster

Part of this bachelor thesis is also a poster which serves to present the work in a clear and engaging way. The poster is included in full A2 size.

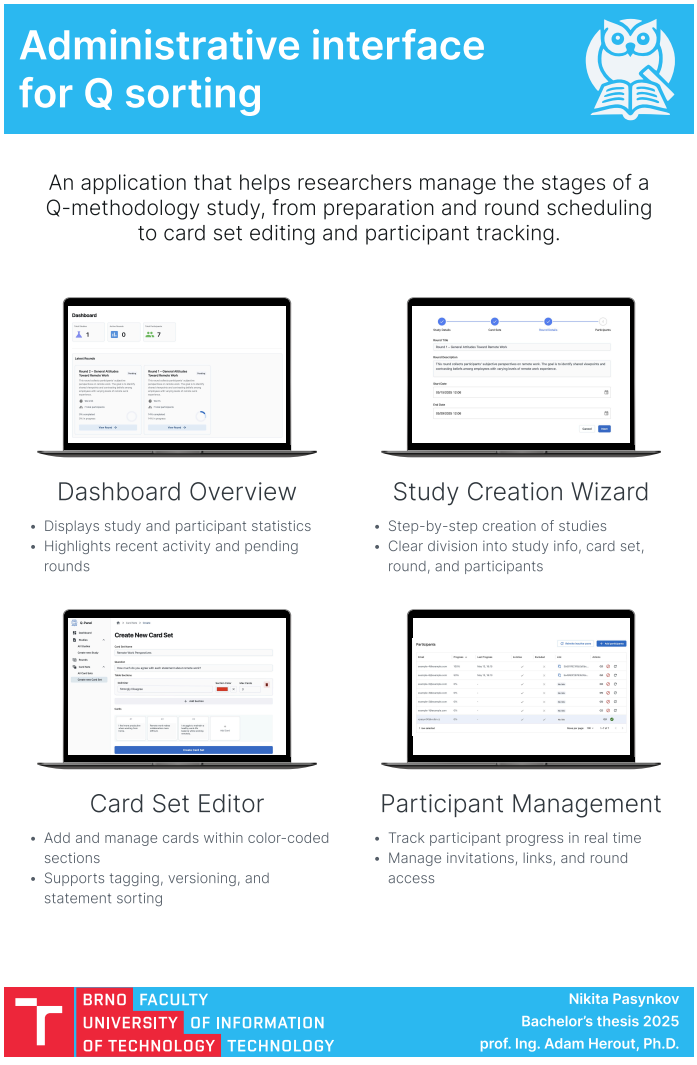

Figure A.1: Poster presenting the Q-methodology admin interface.