

Nazarbayev University
School of Engineering and Digital Sciences
Department of Computer Science
CSCI 409 — Senior Project II

Student Event Ticketing Platform for Nazarbayev University

Final Project Report — Spring 2026

Group 52

Team Members

Alisher Aitkazin, Alisher Seitkazin, Alikhan Serikov,
Daulet Zhumagali, Dosan Saiyn

Project Advisor

Askar Boranbayev

April 2026

1. Executive Summary

Nazarbayev University has a wide range of events, which enrich student life in the form of student clubs, departmental and university bodies. However, the current event management state of affairs at NU relies on fragmented communications such as social media, messaging services and the word of mouth that lead to inefficiencies in event discovery, event registration, and attendance tracking. Students tend to miss out on events because of information not being centralized whereas organizers are not able to control the registration as well as confirming attendance.

The proposed solution discussed in this report is the Student Event Ticketing Platform which is a computing-based solution that will help centralize event management at Nazarbayev university. The platform offers an integrated platform through which students can find out about the upcoming events on campus, register in one action and get digitally verifiable QR based tickets. To the event planners, the system will also provide event creation, instant registration monitoring and QR check-in option. To guarantee conformance with policies, administrators are given the opportunity to moderate event submissions and control the roles of users.

This system was created in two semesters in a sprint-based development that follows the iterative approach. The stack mentions a backend API because it is Go-based with PostgreSQL and Redis, web frontend with React and TypeScript, and an Android mobile application written in Kotlin. The major features implemented are NU email-only access with role-based access control (student, organizer, admin), a centralized event listing with search and filtering, the generation of digital tickets with QR codes, an enforcement with one-ticket-per-student, emailing notifications over SMTP, and a moderation panel. The analytics module is still in development.

Manual testing of the implementation functions was done on all the modules implemented on the platform. Although informal user assessment tests and load testing were not carried out during the duration of the project, the system is seen to be demonstrating a working, combined prototype that can provide the solution to the fundamental issue of disintegrated event management at NU.

2. Introduction

2.1 Problem Statement

During the academic year, Nazarbayev University hosts various events, including academic conferences and guest lectures, as well as student club activities, hackathons, and cultural celebrations. Nonetheless, with this rich event culture, the university does not have a centralized digital platform specific to event discovery, event registration and event attendance management. At the moment, event data are shared via unequal mediums such as Telegram groups, Instagram posts, email notifications, and posters. This fragmentation results in a number of issues: students are not informed of event-related happenings that they might be interested in and organizers cannot accurately predict or control attendance, students can incur multiple registrations and no-show rates are frequent and no stable system is established to check whether attendees are actually attending the event.

2.2 Motivation

This project has been driven by first-hand experience since NU students have experienced these challenges directly. At NU, organizers of events often complain of the inability to reach their target audience, as well as ineffective registration management. In the meantime, students are usually informed about the events when it is too late or they learn it is full when they arrive. These pain points would be resolved through a custom platform specifically for the NU environment, connected to institutional authentication infrastructure, which would not interfere with policy-based institutional approaches to concerns like data privacy and identification of students.

2.3 Proposed Solution

Student Event Ticketing Platform is a complex platform, comprising of web application, Android mobile application and a back-end API. The system enables students to drop any browsing, a centralized events calendar throughout the university, and is capable of participating in events, and, in case they wish to check-in, they receive a digital ticket with an inbuilt QR code. Event creators can engage in creating, managing events, monitoring the registration process, and verifying attendance through QR scanning. The administrators control the site with the help of a moderation panel which ensures that the postings are only real-life events. Rules used in the platform are a one-ticket-per-student restriction, capacity, and NU email-only authentication.

2.4 Report Organization

The rest of this report is set out in the following way. Section 3 establishes the limitations that have an impact on the project. Section 4 reviews background and related work. In section 5, the project approach (architecture, tools and workflows) is outlined. Section 6 is a description of the

implementation of the project in both semesters. Section 7 has assessment methodology and findings. Section 8 has a conclusion which includes a contribution and future work.

3. Identification of Constraints

3.1 Technical Constraints

The system needs to work within the technical boundaries of the university setting. Only NU email addresses (nu.edu.kz domain) can be used to authenticate and this guarantees that only verified members of the university are allowed to use the system, however, it also implies that no external participants are available on the platform. It is implemented as a Docker container-based system, which is portable; however, the deployment is currently restricted to local development environments. The deployment of production would be coordinated with the assistance of NU IT services to provide servers and put up the rules of a firewall and configuration of the domain. Moreover, only Android users can access the mobile application, whereas the web application has a cross-platform reach.

3.2 Economic Constraints

Since it is a scholarly term project, the platform was not created with a specific budget. The tools and services employed are all either open-source (Go, PostgreSQL, Redis, React) or have freetiers. Implementing the production would incur hosting expenses which the university would have to absorb in the future.

3.3 Security and Privacy Constraints

Personal student data is accessed through the platform such as email addresses and past visit history. Based on the NU internal policies and responsible data practices, the system gathers the least amount of data required to operate. The entire communication is based on HTTPS and TLS encryption. The passwords are stored through secure hashing algorithms and authentication on the basis of JWT tokens and short-term access tokens (15 min) and long-term refresh tokens (30 days) to strike a balance between security and usability. Ticket duplication and resale will be prevented with use of the QR code hash.

3.4 Ethical and Social Constraints

Non-relevant criteria in event access should not be a discriminative measure of the system. There are role-wise access control (student, organizer, admin) in that legitimate organizational roles are taken into account and not artificial hierarchies created. Event submissions need to be moderated by administration to avoid abuse but should be done in a transparent manner. The history of attendance of an event is kept in the history as well as a part of the working purposes and reporting and is supposed to be anonymized when it is not needed in active performance.

3.5 Sustainability and Maintainability

It is containerized with Docker making the codebase modular which allows future developers to add on to or maintain the system without onboarding it to a large extent. The backend takes a domain-driven design whereby it has independent modules which include authentication, events, ticketing, payment, notifications, and administration modules. The frontend architecture is based on a component-based structure with React. There are documentation and README files of the backend repositories and frontend repositories.

4. Background and Related Work

4.1 Event Management Platforms

There are a number of business sites that deal with event management and ticketing. Eventbrite is a popular system which facilitates the creation of events and sale of tickets, as well as the management of the registration process and check-in of visitors with the help of QR codes. Although it is comprehensive, Eventbrite is not events-specific and does not support university authentication. Its pricing system also adds expenses that would not be needed within a university setting where most of the events are either free or low-priced. Likewise, platforms at the community level, like Meetup.com are not dedicated to institutional access management or role-based moderation and are designed around community-based organization of events.

4.2 University-Specific Systems

Other universities have integrated internal event management software, usually an extension of their student information systems or learning management systems. Though these usually go hand in hand with proprietary infrastructure and cannot be ported to other institutions. Nazarbayev University had no special event ticketing system yet when this project commenced. The process of event management makes use of manual processes and third-party services that are not linked with the university identity system.

4.3 QR-Based Ticketing and Check-in

QR code technology has emerged as a standard practice in digital ticketing in the transportation, entertainment, and event sector. QR codes represent a two dimensional bar code that is encoded, and through which information is quicker to scan with smartphone cameras. QR codes used in ticketing applications are usually a unique identifier or cryptographic hash, which is authenticated to a database on the server side on entry. This will also avoid ticket replication, it will assist in real time monitoring of the attendees and eliminate the use of physical tickets. The safety of QR-based tickets relies on the individuality and randomness of the data coded.

4.4 Role-Based Access Control

Role-Based Access Control (RBAC) is the stablest management of permissions in multi-user systems. RBAC assigns permissions to a group of roles instead of a specific user, and connects the organisational functionality with the roles. This project provides a three level RBAC design: students may browse the events and purchase tickets, organizers have the same capabilities as students and may also create events and conduct the check-in process, administrators have the same capabilities as organizers and may also moderate events and manage roles of users. This is a hierarchical model that conforms to the principles of RBAC available in access control literature [12].

4.5 Technology Stack Justification

Go programming language [1] was selected to use it in the backend because of its high performance aspects as it is a static typed language with built-in support of concurrency and it can be used in developing HTTP services [4]. PostgreSQL [9] is used as the main database due to its strength, ACID compliance, and proper supremacy of intricate queries. Redis [10] is used as a rate limiting in-memory data store. React with TypeScript [8] was chosen on the frontend due to its component-based structure, good support of type, and rich ecosystem, built with Vite [14] as the build tool. The Android application is coded in Kotlin [13] which is the preferred language used to develop Android applications. Docker [2] offers homogeneous levels of development and deployment to team members.

5. Project Approach

5.1 System Architecture

The system implements a modular monolith architecture backend side, where logically separated domains (authentication, events, ticketing, notifications, administration, analytics) are separated into different Go packages deployed in one deployment package. Such an architecture gives us advantages of domain separation without having all the overheads that microservices architecture implies [11] at the current level of the application scalability. The backend layer is implemented according to the principle of Clean Architecture [7] and is divided into three layers: handler (HTTP requests handling), service (domain business logic), and repository (data accessing).

The backend is accessed via RESTful HTTP API located at the `"/api/v1"` prefix and used by the React web front end and the Kotlin Android app. The API works with JSON bodies, uses JWT tokens for user authentication, and returns standard HTTP response codes for errors.

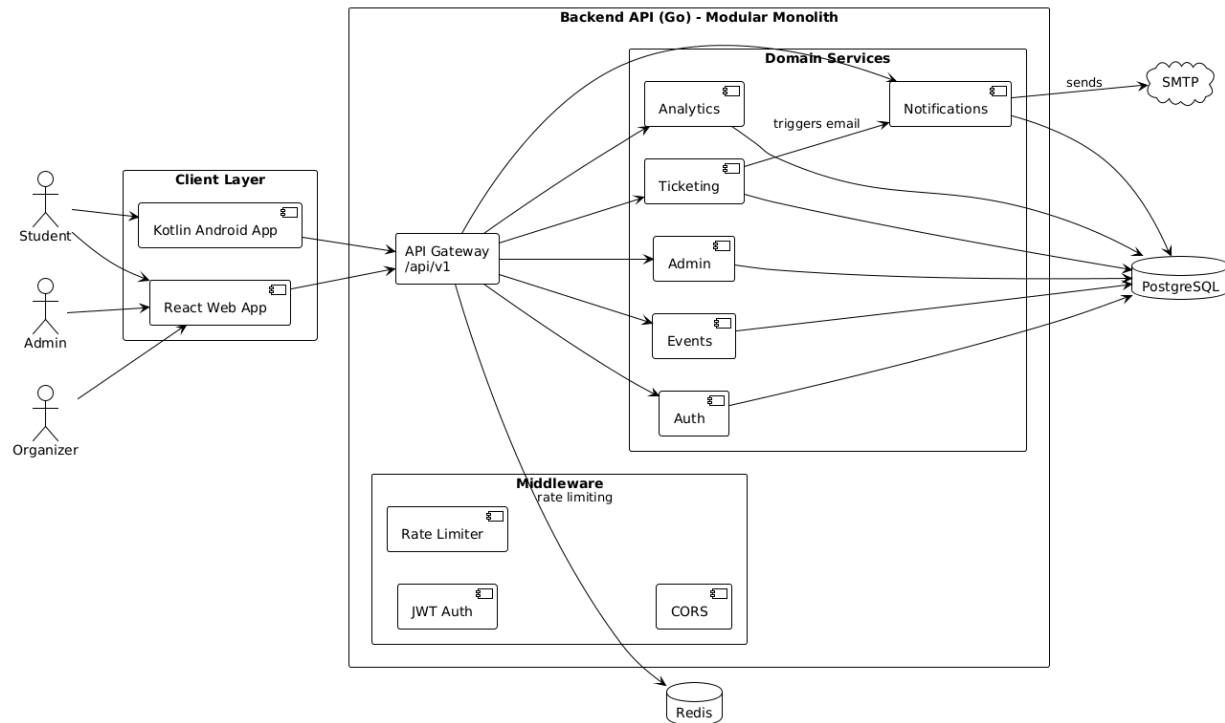


Figure 1. System architecture diagram

5.2 Backend Modules

The backend is structured into the following domain modules, which have handler, service, repository and model layers:

Module	Status	Description
auth	Implemented	User registration, login, JWT token management, NU email validation
events	Implemented	Event CRUD, search/filter with query parameters, moderation status filtering
ticketing	Implemented	Ticket registration, QR code generation, cancellation, QR-based check-in
notifications	Implemented	Email notification queuing and delivery via SMTP
admin	Implemented	Event moderation (approve/reject), user role management
analytics	In Development	Event statistics and reporting (currently returns HTTP 501)

5.3 Frontend Architecture

The web frontend is a single page application created using React 19, TypeScript and Vite. It implements TanStack React Query to manage the state of the server, Zustand to manage the state at the client, (authentication and ticket storage), React Router to navigate the application and AXIOS to communicate with the back-end API. The front end uses role-conscious navigation and route guards which adjust the interface depending on the role of the user with the authentication. Tailwind CSS offers utility-based styling.

The frontend is composed of eleven pages that can be accessed based on user role: public pages (home, login, register, event catalog, event detail), student pages (ticket management with QR display), organizer pages (dashboard, event creation, event editing, QR check-in), administrator pages (moderation queue, user role management).

NU Events

Events

Sign In

NU

Create Account
Register with your NU email

Email

yourname@nu.edu.kz

Password

Minimum 8 characters

Confirm Password

Repeat password

Create Account

Already have an account? [Sign In](#)

Nazarbayev University · Student Event Ticketing Platform · 2026



Welcome Back

Sign in with your NU email

Email

dosan@gmail.com

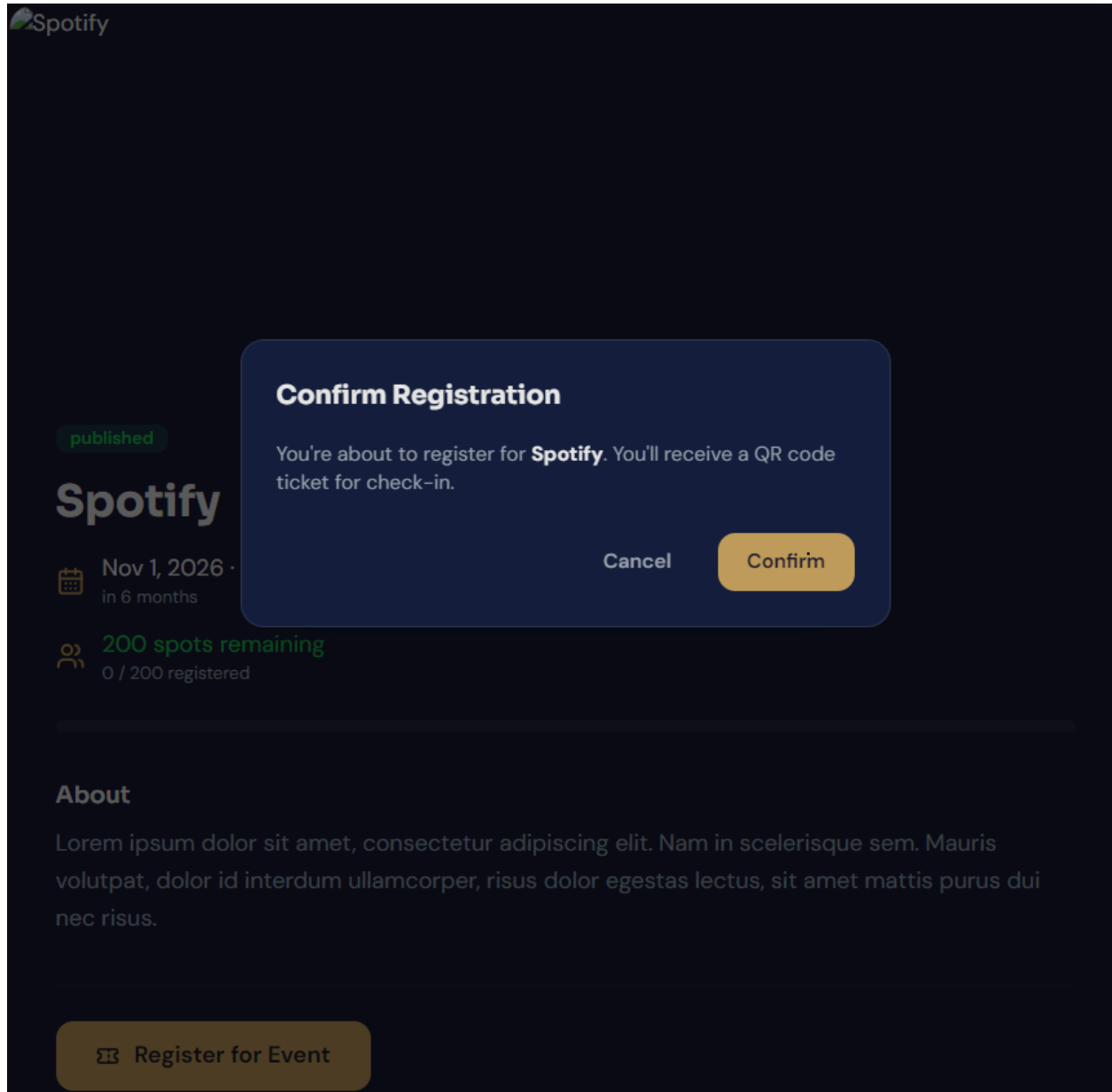
Must be a valid @nu.edu.kz email

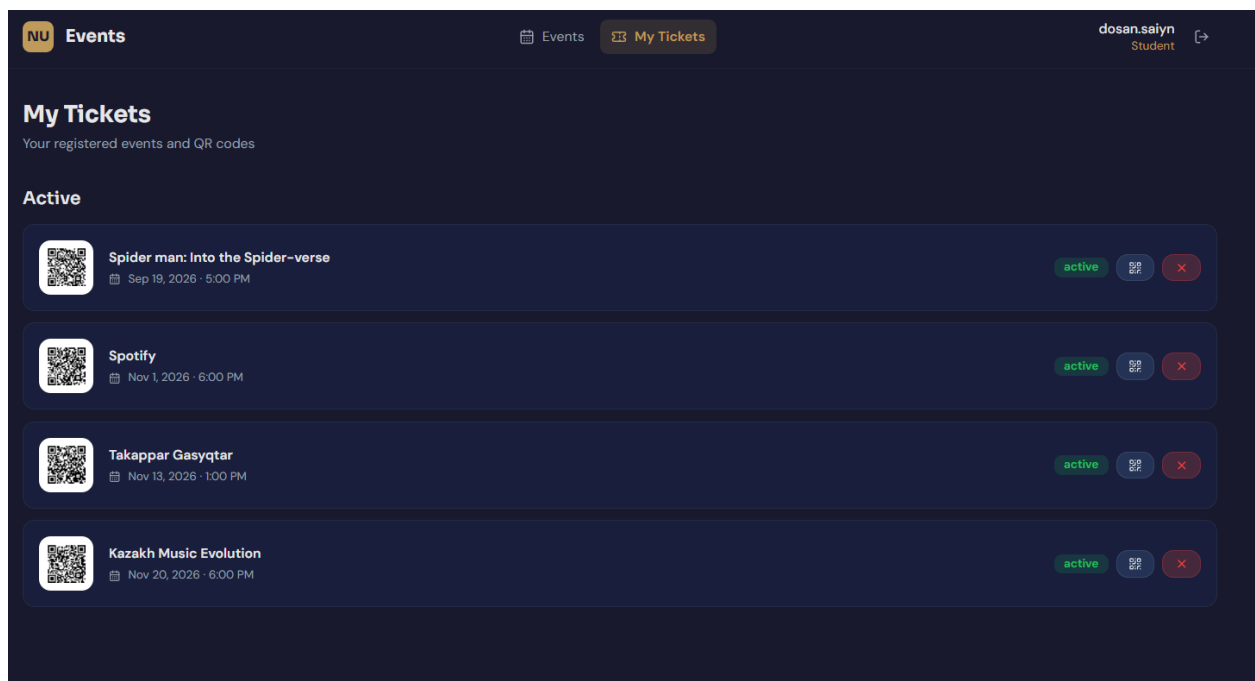
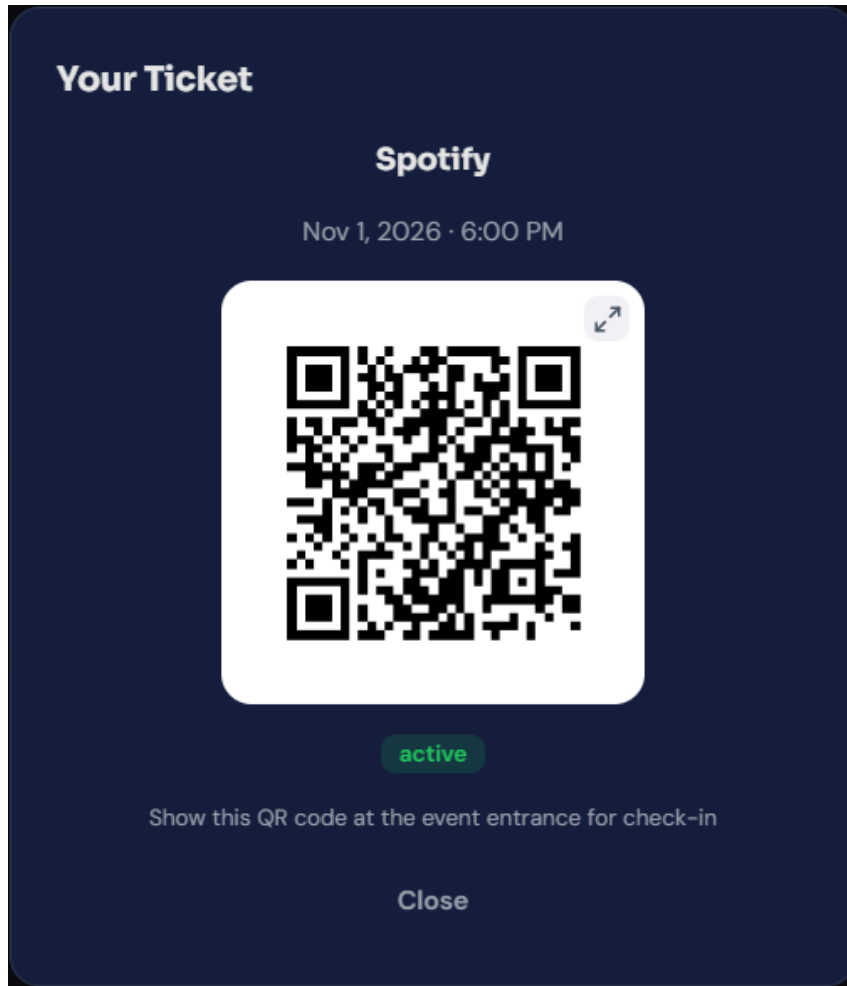
Password

.....

Sign In

Don't have an account? [Register](#)





Events

Events

Sign In

Events

Browse upcoming events at Nazarbayev University

ДД.ММ.ГГГГ

ДД.ММ.ГГГГ

Robot

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in scelerisque sem. Mauris volutpat, dolor id...

Nov 25, 2026 · 6:00 PM

200 / 200 spots

Frequencies

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in scelerisque sem. Mauris volutpat, dolor id...

Nov 16, 2026 · 6:00 PM

200 / 200 spots

Takappap Gasyqtar

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in scelerisque sem. Mauris volutpat, dolor id...

Nov 13, 2026 · 1:00 PM

199 / 200 spots

Shibuya

Spotify

Cozy Korne

← Back to Events

published

Spotify

Nov 1, 2026 · 6:00 PM

in 6 months

199 spots remaining

1 / 200 registered

About

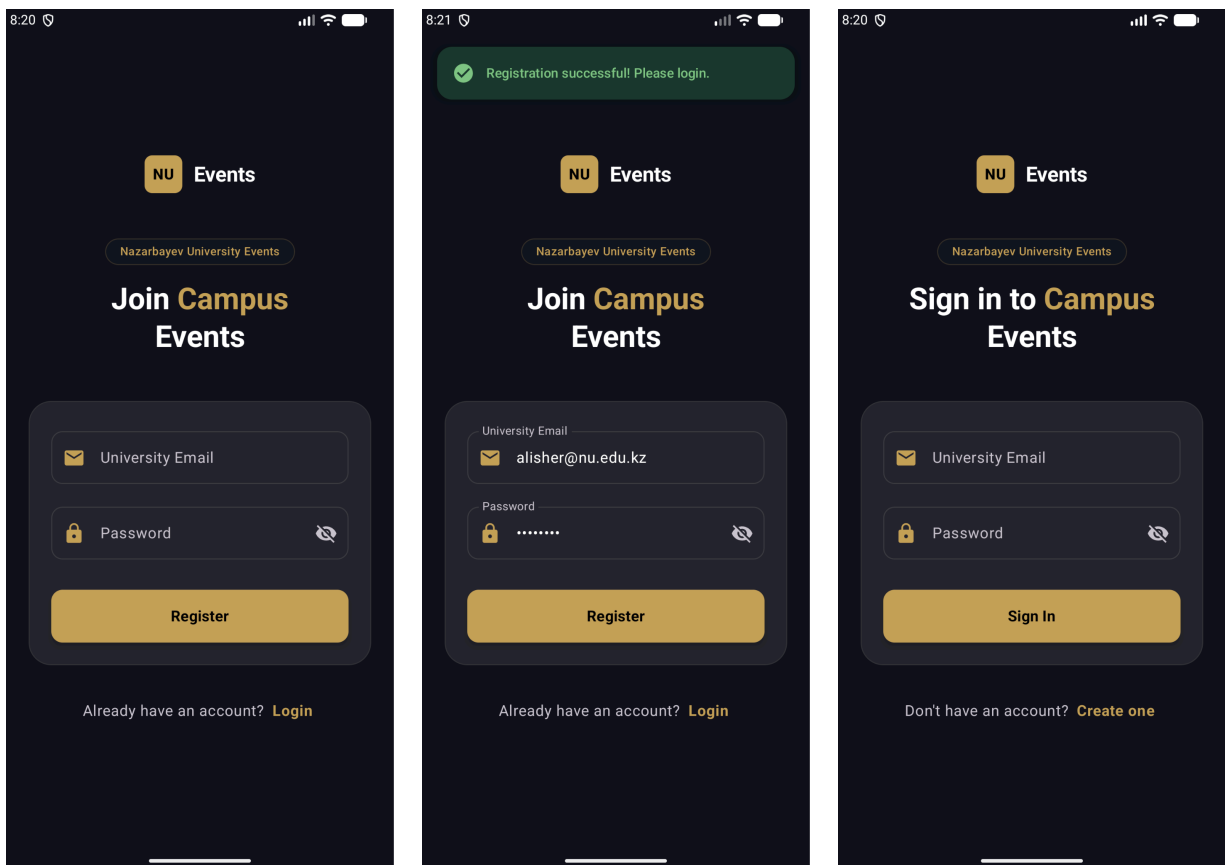
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nam in scelerisque sem. Mauris volutpat, dolor id interdum ullamcorper, risus dolor egestas lectus, sit amet mattis purus dui nec risus.

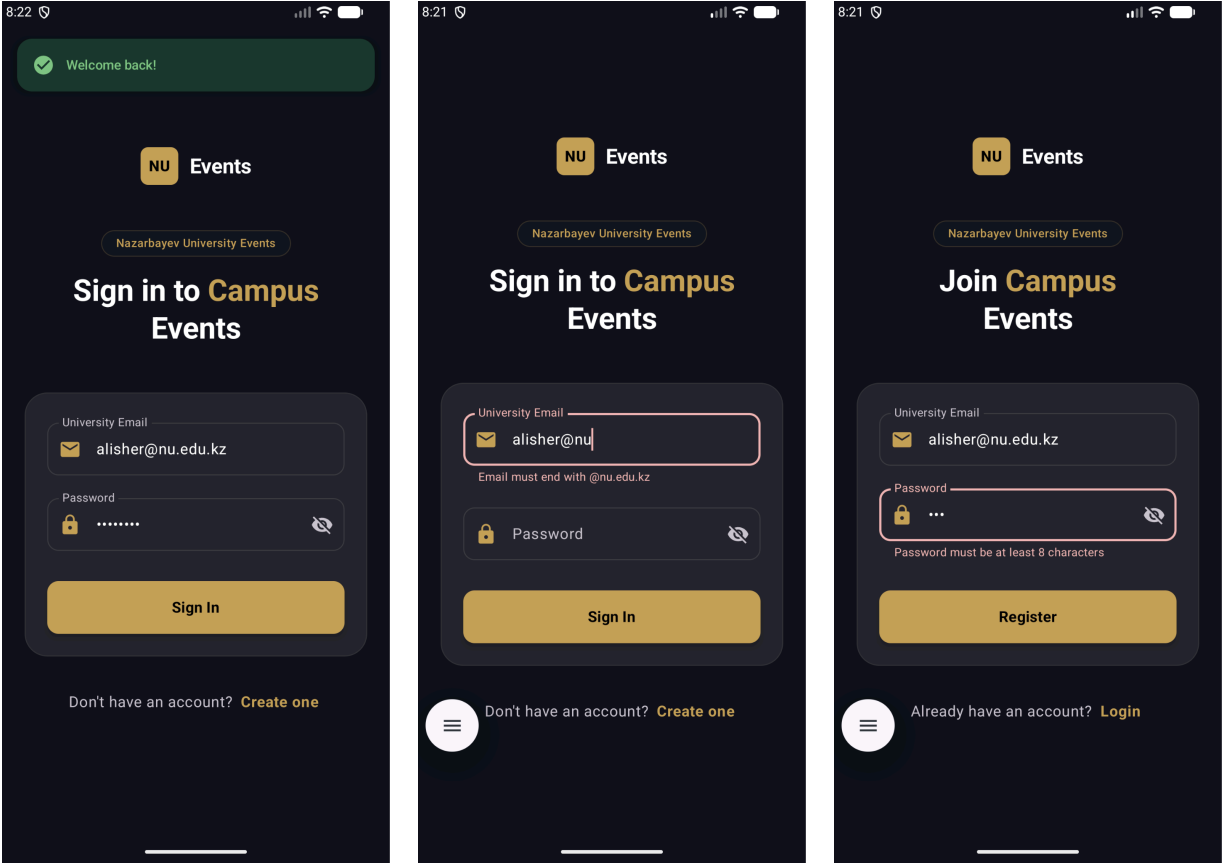
Sign in to Register

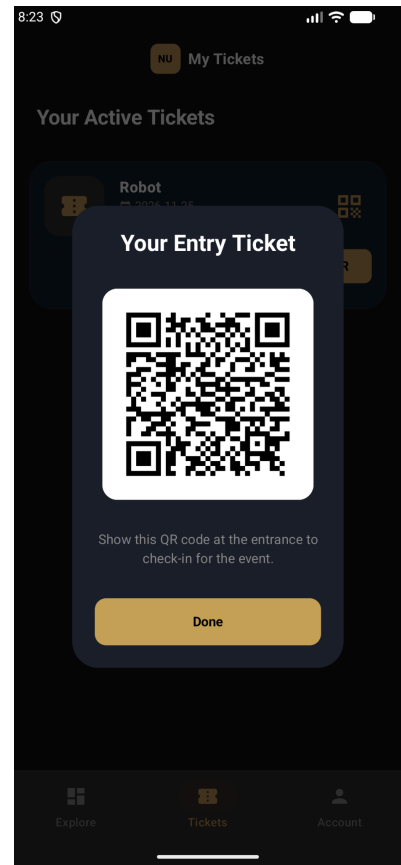
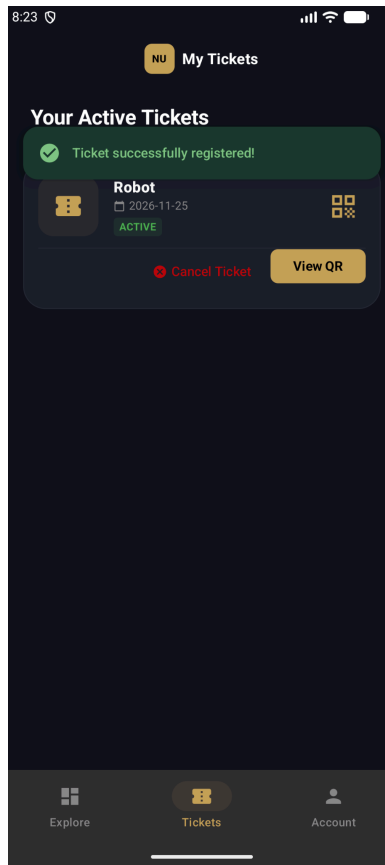
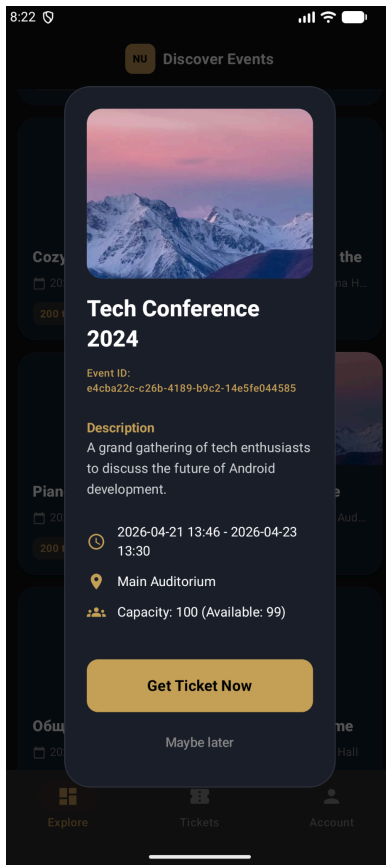
Figure 2: Web Frontend Screenshots

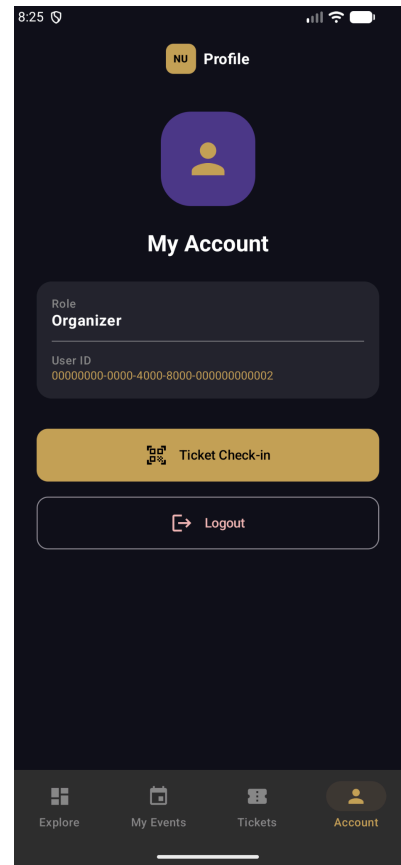
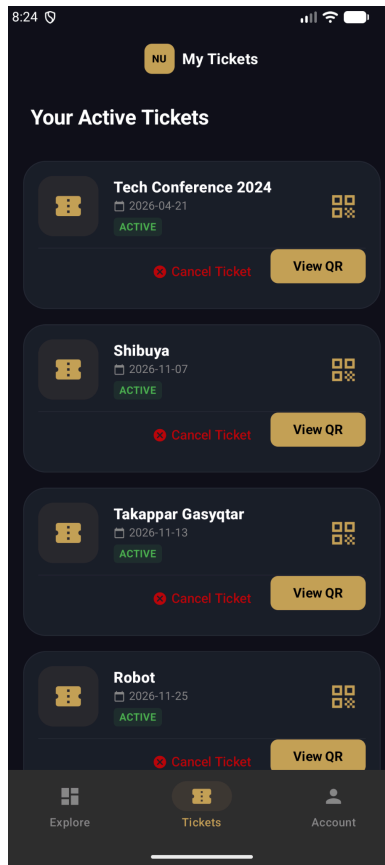
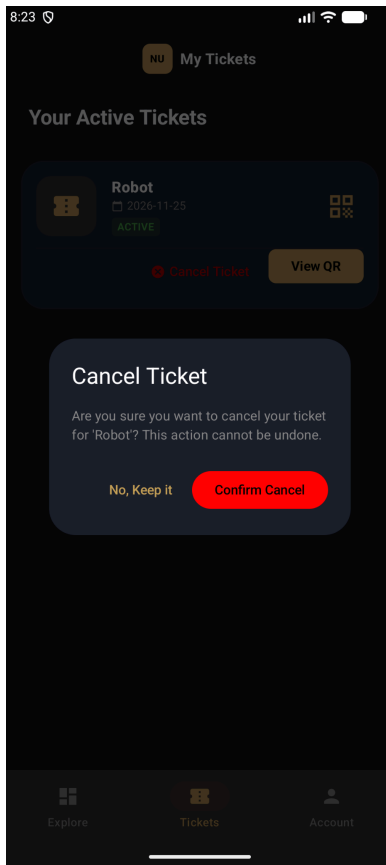
5.4 Mobile Application

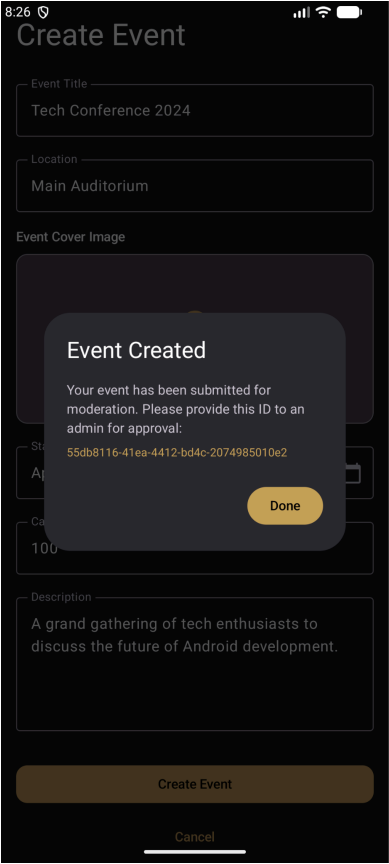
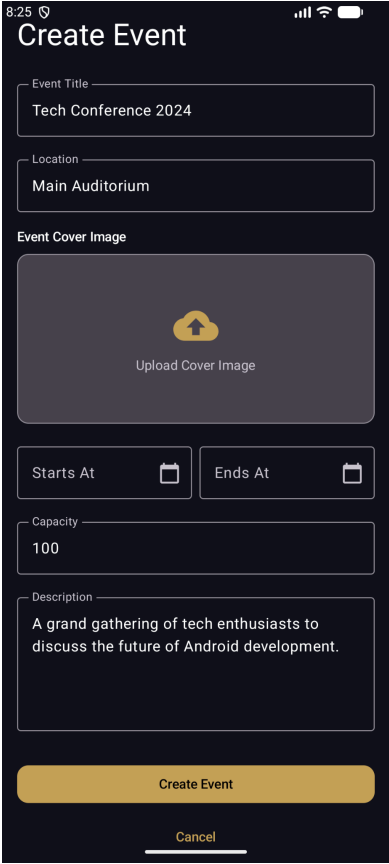
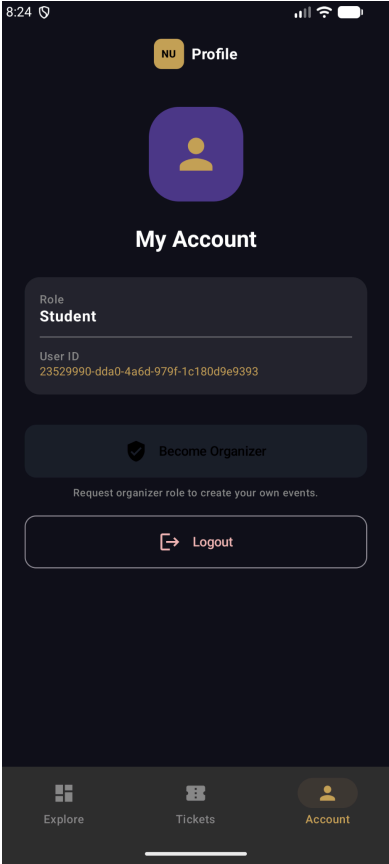
Android mobile app was built in Kotlin and offers the main functionality such as browsing events, registering tickets and displaying QR codes to check-in. This is done using the mobile application in the same endpoint as the web frontend and makes sure the data are compatible across platforms.











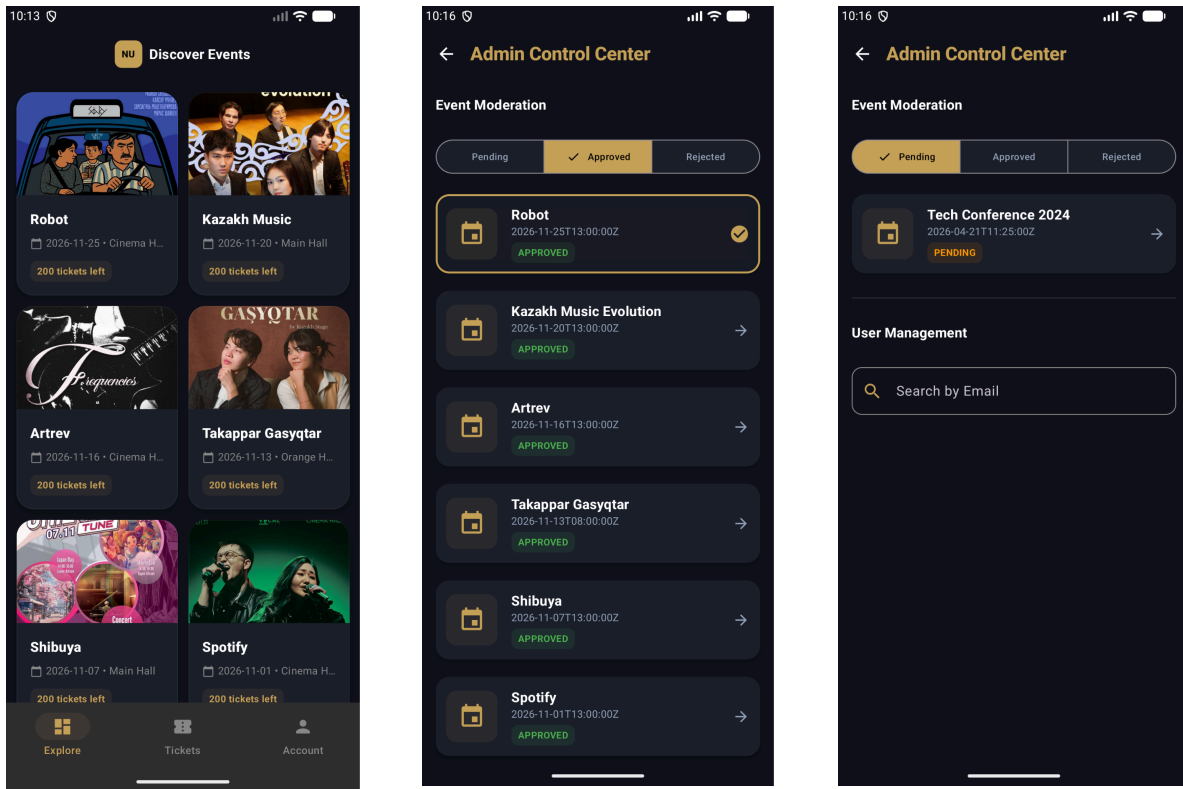


Figure 3. Mobile App Interface

5.5 Database Design

The PostgreSQL database schema consists of user, event, ticket, notification, and refresh token tables. Other important design considerations are UUIDs used as primary keys to avoid enumeration attacks, processing event status (draft, published, cancelled) and moderation status (pending, approved, rejected) in separate columns to decouple the event process and the administrative review process, and cryptographic hashing of QR code data to preclude forgery of tickets.

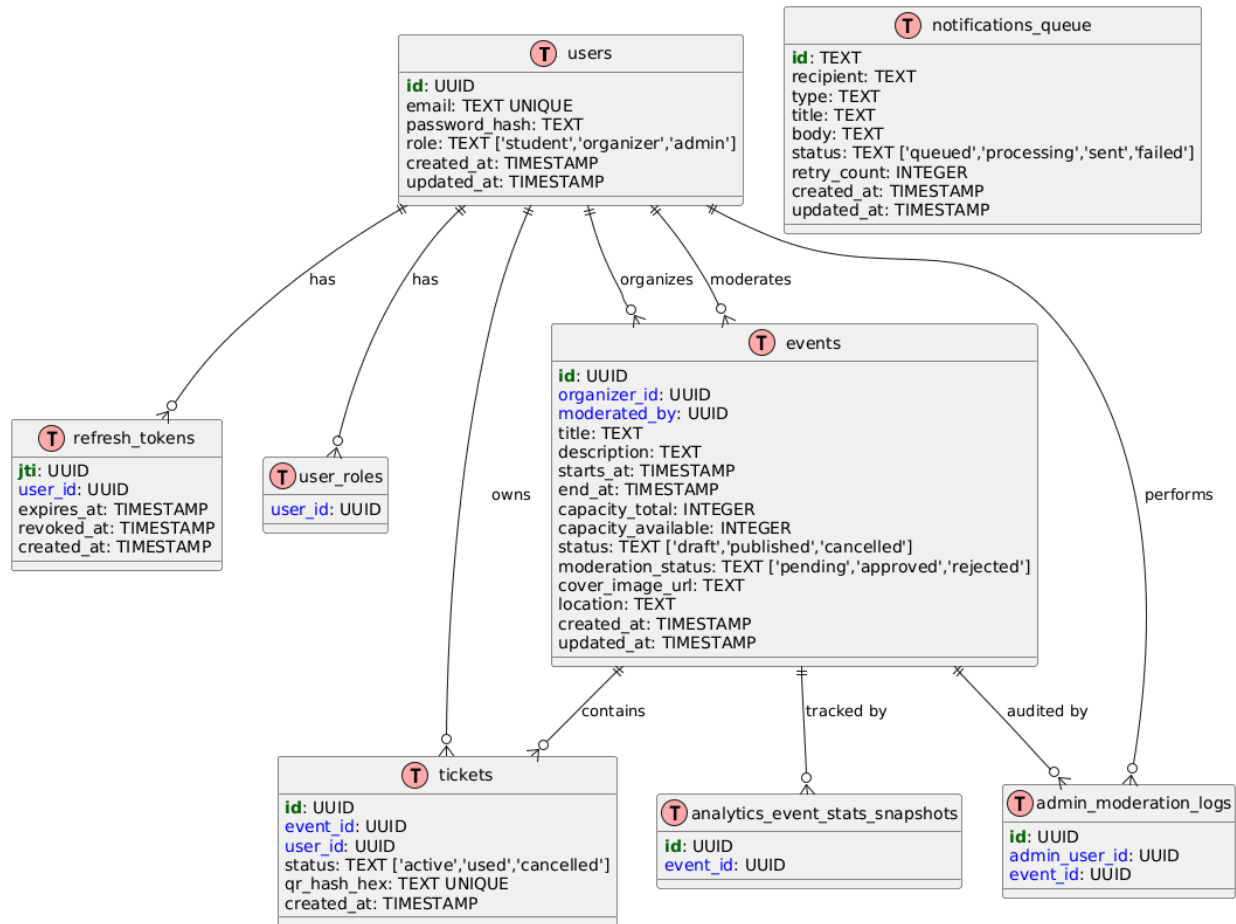


Figure 4. Database ER diagram

5.6 API Design

The API is based on RESTful best practices, and errors are formatted consistently. Any error responses have a standard JSON format with an error code and human understandable message. Each business rule violation has well-defined error codes associated with it (e.g., already or already registered, capacity full, event not approved, etc.) that the frontend uses to show appropriate messages. The rate limiting is implemented through Redis with 120 requests per 60 seconds by the client. As a setup, CORS allows requests to the frontend development server origins.

system		^
GET	/healthz Health check	▼
notifications		^
POST	/notifications/send-email Enqueue outbound email (foundation)	▼
payments		^
POST	/payments/initiate Initiate a payment	▼
POST	/payments/webhook Payment provider webhook	▼
tickets		^
GET	/tickets/my List my tickets	▼
POST	/tickets/register Register a ticket for an event	▼
POST	/tickets/use Mark ticket as used	▼
POST	/tickets/{id}/cancel Cancel a ticket	▼
admin		^
POST	/admin/events/{id}/moderate Moderate event visibility (admin only)	▼
GET	/admin/moderation-logs List moderation audit logs (admin only)	▼
PATCH	/admin/users/{id}/role Set user role (admin only)	▼
analytics		^
GET	/analytics/events/stats Event statistics	▼
auth		^
POST	/auth/login Login and get tokens	▼
PATCH	/auth/me/roles Request organizer role (pending until admin approves)	▼
POST	/auth/refresh Refresh access token using refresh token	▼
POST	/auth/register Register a new user	▼
events		^
GET	/events List events	▼
POST	/events Create an event	▼
GET	/events/{id} Get event by ID	▼



Figure 5: Swagger UI displaying all available API endpoints under the /api/v1 prefix, including authentication, event management, ticketing, and administrative operations.

5.7 Third-Party Components

Component	Purpose	Integration Method
PostgreSQL	Primary relational database	Go pgx driver, Docker container
Redis	Rate limiting and caching	Go redis client, Docker container
Chi Router	HTTP request routing (Go)	Go module dependency
JWT (Go-JWT)	Token-based authentication	Go module dependency
Swagger (swag)	API documentation generation	Code annotations, auto-generated
React + Vite	Web frontend framework	Node.js development server
TanStack React Query	Server state management	npm dependency
Zustand	Client-side state management	npm dependency
Docker	Containerized deployment	docker-compose orchestration
SMTP Server	Email notification delivery	Backend notification module

5.8 Team Roles and Coordination

Team Member	Role	Responsibilities
Alisher Aitkazin	Database & Infrastructure	Go API development, database logic, authentication, ticketing
Dosan Saiyn	PM & QA Engineer	React web app, interface elements, API.
Alisher Seitzkazin	Mobile Developer	Android mobile interface Kotlin Android application.
Daulet Zhumagali	Backend Lead	PostgreSQL schema, Docker deployment, Redis set up.
Alikhan Serikov	Frontend Lead	Sprint planning, coordination, testing, documentation

The team planned the sprints and tracked the tasks in Trello, managed the source code in GitHub and reviewed it, and communicated daily in Telegram. The development was an iterative sprint based development with frequent review meetings.

5.9 Use Case Summary

The main scenarios of the system are: (1) a student creates an account with an NU email, views events, registers an event, receives a QR ticket and presents it at the check-in; (2) an organizer creates an event, which goes to the pending moderation state, and once approved is visible in the catalog, after which the organizer manages registrations and presents it at the check-in; (3) an administrator views

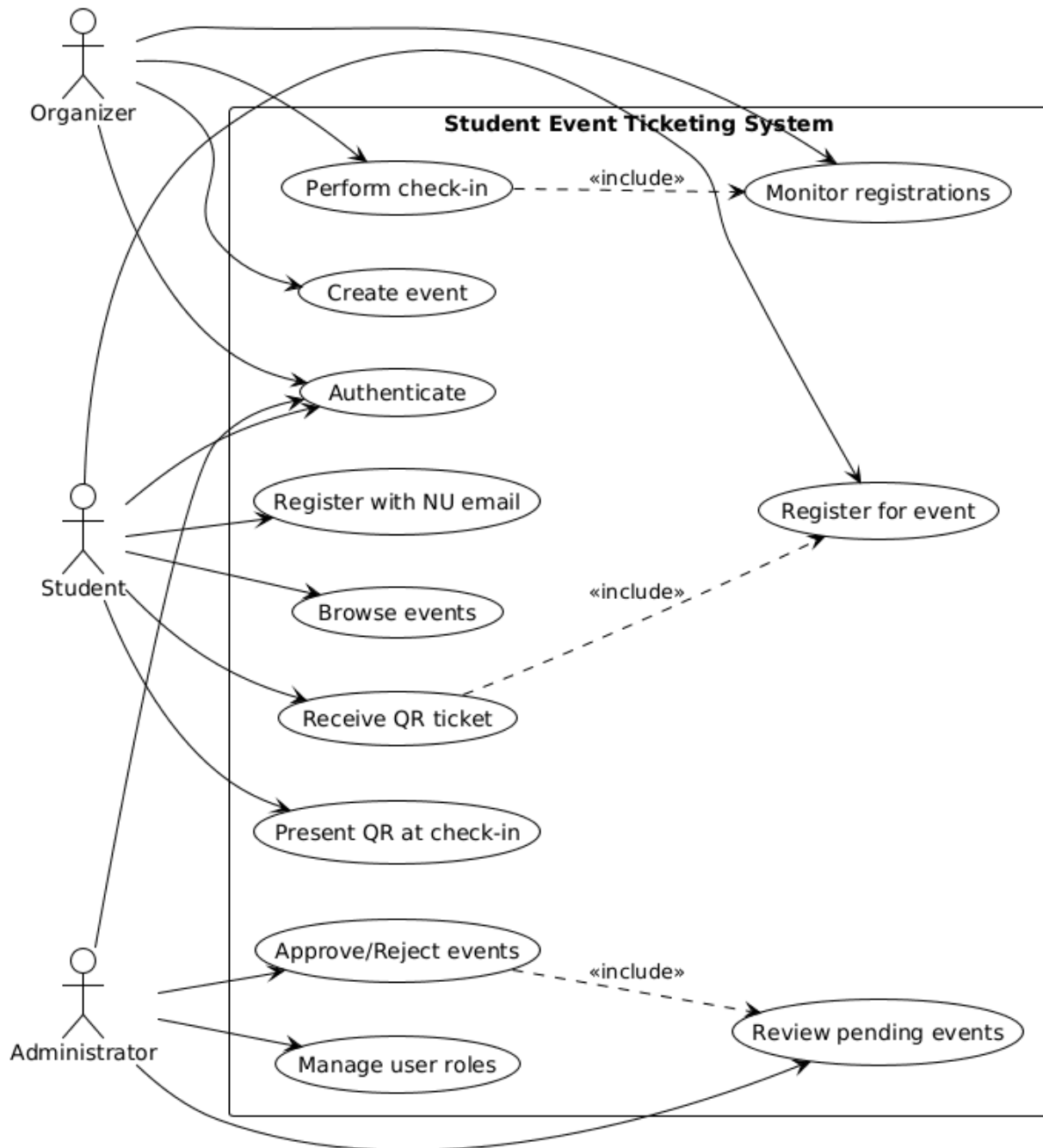


Figure 6: Use Case Diagram

6. Project Execution

6.1 Fall 2025 (Semester 1)

In the first semester, the efforts were directed at gathering the requirements, technology selection, and prototyping. The group analyzed the existing event management systems, gathered requirements (both functional and non-functional), chose a technology stack, and started designing and building prototypes for the mobile application, along with the first versions of its frontend interface. At that point, however, no work had been done to integrate those features with the backend services and databases. Thus, at the end of the semester, we came up with the requirements specification and project plan.

6.2 Spring 2026 (Semester 2)

The second semester was dedicated to full implementation, integration, and testing. Development proceeded in five phases aligned with the project plan:

Phase 1: Foundation and Core Integration (January–February)

The stage involved the creation of the authentication mechanism using the NU email check along with the session management using JWTs. Role-based access control was developed. The database structure for event management, ticket management, and user management was created. The front end code was also modified to facilitate the process.

Phase 2: Digital Ticketing and QR System (February–March)

Ticket QR code generation was done in the backend and returned a Base64 encoded image file type PNG for the QR code. Ticket per student limitation was also done from the database side. Mobile app was upgraded to scan QR codes as well. A frontend ticket page that could show the QR code and even go into full screen was created.

Phase 3: Notifications (March)

An SMTP server was used to send registration confirmations and updates on the event through email notification system. The checkout flow was connected to the payment API in the frontend.

Phase 4: Organizer Dashboard and Admin Panel (March–April)

Organizer interface was created which consists of event creation, event editing and event statistics. Admin panel interface was also developed which comprises of approving and rejecting the event with reason being given by the approver, and the roles of the users.

Phase 5: Testing and Documentation (April)

Functional testing was performed manually in all the modules. The team tested every functional requirement by end-to-end testing user flows of the web frontend, mobile application, and backend API. The documentation was also updated, such as API documentation through Swagger and project README files.

6.3 Deviations from the Original Plan

The project had several aspects that were not as per the plan. Although some of the real-time capacity enforcement system (FR-8) was implemented with database-level constraints, it was not applied as a complete real-time system with WebSocket notifications. Organizers real-time statistics dashboard (FR-10) was not available, since the analytic module behind it is still being developed. The performance testing of 1,000 simultaneous users was not done because of the time limitations and local-only environment of deployment. Also, the mobile application is written in Kotlin, instead of a native Android development framework, which was initially intended to be Flutter, because the mobile developer is more proficient with Kotlin than Flutter.

6.4 Challenges Encountered

A number of challenges were experienced when implementing the project. First, there was the issue of the scope creep where new functions and cases kept on rising. Another challenge was the integration of the front end, backend, and mobile applications that necessitated adequate management of the API contract. The format of response would be different at times than expected, and it would be a time-consuming process to debug. In addition, coordinating the activities within the team was not an easy task due to varying timetables of each of the five members.

7. Evaluation

7.1 Evaluation Approach

The student event ticketing system was evaluated by looking at the answer to the question posed in the introduction, which is the lack of a single platform to control event discoveries, registrations, and event attendance at NU. Evaluation was conducted manually wherein tests were carried out on every module to establish whether it was suitable to the functional requirements.

7.2 Functional Validation

End-to-end user flow testing was done on each functional requirement. The validation of each requirement is summarized in the following table:

ID	Requirement	Status	Notes
FR-1	NU email authentication	Validated	Registration does not accept non-nu.edu.kz emails; login sends JWT tokens.
FR-2	Role-based access control	Validated	Three roles implemented; route guards at the frontend; middleware at the backend.

FR-3	Centralized event catalog	Validated	Web and Android-app accessible events.
FR-4	Search and filter events	Validated	Text search, date range filtering, pagination implemented
FR-5	Student event registration	Validated	Students enroll through web or mobile; receipt of confirmation.
FR-6	QR code digital ticket	Validated	QR PNG created and shown; can be viewed in full-screen.
FR-7	One ticket per student	Validated	Authenticate Duplicate registration 409 (already registered)
FR-8	Capacity enforcement	Partial	Enforced at the database level; notifications are not in real-time.
FR-9	Organizer event management	Validated	Modify, create, cancel events via web interface.
FR-10	Real-time statistics	Not Implemented	Developing analytics module (501)
FR-11	Notifications	Validated	SMTP email notifications working.
FR-12	Paid ticket flow	Not implemented	Out of scope
FR-13	Admin moderation	Validated	Accept/decline events; assign user roles.

7.3 System Testing

Testing of the web frontend, mobile app, and API was done manually. There were happy paths (sign-up, create events and issue tickets) and error paths (duplicates, no available places, token expiration and invalid input data). The backend API was run through the use of curl commands and through Swagger UI. Frontend was tested on the latest browsers like Chrome and Safari, and the mobile app was tested on Android phones.

Automated testing, including unit testing and integration testing, was not carried out in accordance with the project timeline. Scalability was required to be tested in terms of load testing, but it was not done because the solution works only locally inside the Docker container.

7.4 User Evaluation

There was no user evaluation exercise that involved the use of questionnaires in the time available for the completion of this project. The system was shown to the project supervisor and other team members, who gave some form of informal feedback on its usability. In future

studies, user evaluation is recommended for determining usability and the overall usefulness of the platform.

7.5 Limitations of Current Evaluation

There are several shortcomings in this particular evaluation process. First, the inability to carry out automatic tests implies that there is no way to detect any regressions other than manually. Second, the lack of load tests implies that the performance and behavior of the application during simultaneous use cannot be validated. Third, the lack of any kind of user testing implies that no empirical evidence exists regarding the usability and user satisfaction levels.

8. Conclusion and Possible Future Work

8.1 Summary of Contributions

This was a project that designed, implemented, and tested a Student Event Ticketing Platform at Nazarbayev University. This system is an integrated approach in event discovery, registration and electronic ticketing, which was nonexistent with the prior, fragmented attempts to manage events at Nazarbayev University. The system comprises of NU email-based authentication, role-based authorization, event search engine, QR-based electronic tickets with one ticket one-student verification, payment processing, email notifications, event management by organizers and moderation administration. This platform is a working prototype that consists of a Go server-side API, a React web client, and Kotlin mobile app, which are packaged with Docker.

8.2 Future Work

A number of recommendations regarding further development have been outlined:

Completion of the analytics module to give information about registration statistics, revenue, and attendance trends. The introduction of real-time effects by connecting with WebSockets to give live updates on the capacity of the venue and notifications to register users. Creation of the mobile application of the iOS devices to supplement the current application of Android. Implementation of the project on the cloud or NU based servers, which involves establishing the domain, acquiring the SSL certificate, and automation of the procedure of backing up the database. User evaluations to get structured feedbacks by the students and organizers. Automated tests that will be implemented to test units, integrations and entire applications. Testing to ensure that it can support 1,000 users simultaneously.

9. References

- [1] Bodner, J. (2021). Learning Go: An idiomatic approach to real-world Go programming. O'Reilly Media.
- [2] Docker. (n.d.). Docker documentation. Retrieved September 21, 2025, from <https://docs.docker.com/>
- [3] Eventbrite. (n.d.). Eventbrite platform. Retrieved March 2026, from <https://www.eventbrite.com/>
- [4] Go Programming Language. (n.d.). The Go programming language documentation. Retrieved March 2026, from <https://go.dev/doc/>
- [5] Kim, G., Behr, K., & Spafford, G. (2013). The Phoenix project: A novel about IT, DevOps, and helping your business win. IT Revolution Press.
- [6] Kleppmann, M. (2017). Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media.
- [7] Martin, R. C. (2017). Clean architecture: A craftsman's guide to software structure and design. Prentice Hall.
- [8] Meta Platforms. (n.d.). React documentation. Retrieved March 2026, from <https://react.dev/>
- [9] PostgreSQL Global Development Group. (n.d.). PostgreSQL documentation. Retrieved September 21, 2025, from <https://www.postgresql.org/docs/>
- [10] Redis Ltd. (n.d.). Redis documentation. Retrieved March 2026, from <https://redis.io/docs/>
- [11] Richardson, C. (2018). Microservices patterns: With examples in Java. Manning Publications.
- [12] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., & Youman, C. E. (1996). Role-based access control models. IEEE Computer, 29(2), 38–47.
- [13] JetBrains. (n.d.). Kotlin documentation. Retrieved March 2026, from <https://kotlinlang.org/docs/>
- [14] Vite. (n.d.). Vite: Next generation frontend tooling. Retrieved March 2026, from <https://vite.dev/>

Appendix: API Endpoint Reference

Method	Path	Auth	Role	Description
POST	/auth/register	No	—	Register with NU email
POST	/auth/login	No	—	Login, receive JWT tokens
POST	/auth/refresh	No	—	Refresh token pair
GET	/events/	No	—	List approved events (search, filter, paginate)
GET	/events/{id}	No	—	Get event detail (approved only)
POST	/events/	Bearer	organizer, admin	Create event (starts as pending)
PUT	/events/{id}	No	—	Update event
DELETE	/events/{id}	No	—	Delete event
POST	/tickets/register	Bearer	student	Register for event, receive QR
POST	/tickets/{id}/cancel	Bearer	student	Cancel own ticket
POST	/tickets/use	Bearer	organizer, admin	Check-in by QR hash
POST	/notifications/send-email	No	—	Enqueue email
GET	/analytics/events/stats	Bearer	any role	Event statistics (501)
POST	/admin/events/{id}/moderate	Bearer	admin	Approve or reject event
PATCH	/admin/users/{id}/role	Bearer	admin	Change user role