VILNIUS UNIVERSITY
KAUNAS FACULTY

INSTITUTE OF SOCIAL SCIENCES AND APPLIED INFORMATICS

Study Programme Information Systems and Cybersecurity

State code 6121BX003

DZMITRY PAPKOU

BACHELOR'S THESIS

"EVENT GOOSE": EVENT ORGANISATION AND REGISTRATION PLATFORM

Kaunas, 2025

VILNIUS UNIVERSITY
KAUNAS FACULTY

INSTITUTE OF SOCIAL SCIENCES AND APPLIED INFORMATICS

DZMITRY PAPKOU

BACHELOR'S THESIS

"EVENT GOOSE": EVENT ORGANISATION AND REGISTRATION PLATFORM

Allowed to defend _____

Bachelor student _____
(signature)

Scientific advisor _____
(signature)

_____
(scientific degree of the advisor, scientific pedagogical name, name and surname)

Thesis submitted on _____

Registration No: _____

Kaunas, 2025

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

API	Application Programming Interface
AWS	Amazon Web Services
CI/CD	Continuous Integration and Continuous Delivery
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
DevOps	Development and Operations
DNS	Domain Name System
DOM	Document Object Model
DTO	Data Transfer Object
EC2	Elastic Compute Cloud
ERC	Elastic Container Registry
FK	Foreign Key
GB	Gigabytes
GDPR	General Data Protection Regulation
HTTPS	Hypertext Transfer Protocol Secure
ID	Unique Identifier
IS	Information System
JSON	JavaScript Object Notation
JPA	Java Persistence API
NAT	Network Address Translation
NoSQL	Not Only SQL (Structured Query Language)
PK	Primary Key
RAM	Random Access Memory
S3	Simple Storage Service
SES	Simple Email Service
UML	Unified Modeling Language
UUID	Universally Unique Identifier
VPC	Virtual Private Cloud
VS Code	Visual Studio Code

PAPKOU, Dzmitry. (2025) *“Event Goose”: Event Organization and Registration Platform.* Bachelor’s Thesis. Kaunas: Vilnius University Kaunas Faculty. 61 pages

SUMMARY

The document presents “Event Goose” an event organization and registration platform designed to simplify event management for individuals and organizations. The thesis identifies challenges in the event planning industry, such as insufficient personalization, fragmented services, and data security concerns. “Event Goose” as a concept of a new event planning platform, offers an integrated solution for event discovery, creation, promotion, registration, and payment processing, all under a single-page application. The aim of the thesis is to design a scalable, secure, and engaging event management platform, meeting the latest standards of usability and data security. The objectives include conducting market analysis, designing a web interface, integrating core functionalities, ensuring GDPR compliance, and evaluating the system via manual testing and smoke testing. The methodology employs Agile principles and DevOps practices, utilizing tools like AWS for hosting, PostgreSQL for data storage, and Java with Spring Framework for backend development. The frontend is developed using React and TypeScript for dynamic user interactions. Key results include the architectural design of the system, a robust database schema, and an implemented prototype capable of processing event registration and management. Conclusions underline the platform's potential to enhance user engagement, ensure data protection, and integrate diverse event planning functionalities into a single service.

Keywords: Event management, event planning, scalability, GDPR compliance, personalization, Agile, DevOps, freemium business model, AWS, PostgreSQL.

PAPKOU, Dzmitry. (2025) *„Event Goose" – renginių planavimo ir registracijos platforma.* Bakalauro baigiamasis darbas. Kaunas: Vilnius universitetas, Kauno fakultetas 61 psl.

SANTRAUKA

Baigiamajame darbe aprašoma „Event Goose" – renginių organizavimo ir registracijos platforma, skirta supaprastinti renginių valdymą individualiems vartotojams ir organizacijoms. Baigiamajame darbe identifikuojami renginių planavimo industrijos iššūkiai, tokie kaip nepakankama personalizacija, nesuderintos paslaugos ir duomenų saugumo problemos. „Event Goose", kaip naujos renginių planavimo platformos koncepcija, siūlo integruotą sprendimą renginių paieškai, kūrimui, reklamai, registracijai ir mokėjimų apdorojimui, visa tai pateikiant vieno puslapio programoje. Baigiamojo darbo tikslas – sukurti lanksčią, kintamą, saugią ir įtraukiančią renginių valdymo platformą, atitinkančią naujausius naudojimo ir duomenų apsaugos standartus. Uždaviniai apima rinkos analizės atlikimą, internetinės sąsajos kūrimą, pagrindinių funkcijų integravimą, BDAR laikymosi užtikrinimą bei sistemos vertinimą atliekant rankinį ir pradinį testavimą. Baigiamajame darbe taikomi Agile principai ir DevOps praktikos, naudojant tokius įrankius (angl. *backend*) kūrimui. Išorinė sistema (angl. *frontend*) sukurta naudojant dinamiškoms vartotojo sąsajoms skirtas priemones React ir TypeScript. Pagrindiniai rezultatai apima sistemos architektūrinį dizainą, patikimą duomenų bazės schemą ir sukurtą prototipą, galintį apdoroti renginių registraciją ir valdymą. Išvados pabrėžia platformos potencialą didinti vartotojų įsitraukimą, užtikrinti duomenų apsaugą ir integruoti įvairias renginių planavimo funkcijas į vieną paslaugą.

Raktažodžiai: renginių valdymas, renginių planavimas, lankstumas, kintamumas, BDAR, personalizacija, Agile, DevOps, freemium verslo modelis, AWS, PostgreSQL.

INTRODUCTION

The event's industry has undergone not a small amount of hardship in recent years. Especially during the Covid-19 pandemic arisen in 2019 when most countries of the world-imposed restrictions on gatherings and social events for their own citizens and tourists. Since then, many businesses and organizations related to tourism and events have been forced to close due to the lack of a long-term cash cushion and unclear prospects for their existence.

However, nowadays, the situation is normalized and more and more people after a long break seek to spend their leisure time away from home, trying to get as many positive impressions as possible from reconnect and engage in events that could provide meaningful experiences. Nevertheless, many individuals and organizations face difficulties in finding and organizing events that are both accessible and impactful, which usually lead to missing opportunities and diminished enthusiasm for planning process.

This paperwork presents "Event Goose" an event management system designed to serve as web platform to create, find, and participate in events, by simplifying the entire event lifecycle, from organization to registration, providing users with the tools they need to bring their ideas to life and connect with others who are interested in such activities.

Main problem: Individuals and organizations often face the complicated task of harmonizing their desires with number of available offers at the market, which after all leads to an overwhelming and often unsatisfactory planning process.

Proposed solution: "Event Goose" is a system for solving these problems, offering all possible solutions for a future creating, managing, and discovering events and integrating various elements of events planning into a single platform.

Aim: To design and develop a complex and intuitive event planning and registration system. To achieve specified aim of the thesis, the following **objectives** must be met:

1. Conduct analysis of the market to define existing event planning and registration solutions;
2. Create a system that integrates planning options such as event search, creation, promotion, and participant registrations;
3. Implement a web interface that will simplify the planning process and enhances user interactions with the system components;
4. Develop an automated pipeline with cloud services integration to validate the system's readiness for production deployment;
5. Implement payment system integration to support payment transactions for users and businesses.

Research methods: The main research methodology will be market analysis by comparing similar solutions to assess the prospects of the system being developed and to identify features that are not available in the current market solutions.

Methods of designing the information system: For design of the system used methods focused on the Agile methodology and principles of DevOps. The system architecture is presented by UML to visually represent use cases and both functional and non-functional requirements to the system. Tools such as VS Code utilize for coding, while Postman and Firefox Developer Tools are used for testing during development, and the GitLab used as the version control system.

Difficulties and limitations of the thesis: The main challenges faced are short time limits for system developing and testing, and limitations of the available third-party APIs integrations for not production launched application.

Justification of logical structure of the thesis: The logical structure of this paperwork is designed to follow a logical structure described in the methodological requirements for the final thesis from conceptualization to implementation.

The most significant literary sources used: Various literary sources were used in the paper, including information from the Internet, books and scientific articles. The thesis is written using the methodological requirements provided by the university [26].

Data on the implementation of the developed system: At the time of writing the final part of the paperwork, the implementation of the system has not been done fully, most focus was applied to the backend of the system as the most important part of the project. The system project was deployed and accessible in the AWS Cloud under “https://www.traveldrago.com” domain name.

The structure and the scope of the thesis: The first part of the thesis is divided into 6 main chapters: Introduction, Analysis, Technical Task, Project Design, Information System Project Implementation, and Conclusions. The document itself consists of 59 pages (excluding the list of references and annexes), 11 diagrams and 23 table.

1. ANALYSIS

The purpose of this section is to analyse the current event management information system solutions market and based on the analysis of such well-known organizations in that business area as “Eventbrite”, “Meetup” and “Ticketmaster” for organizing and managing events. By assessing these platforms this study will identify market needs, select unique features of each analysed system that may be useful in developing a new information solution, and more important to evaluate the feasibility of a new information system. The analysis will also consider both the limitations of existing solutions and best practices that will be used to identify the unique value that in the development of “Event Goose” can provide.

1.1. Overview of event planning industry challenges

First, let’s look at event planning industry as a whole and at variety of challenges it is faces that affect both the organizers who create events and the participants who seek for experiences. Therefore, to understand the event planning industry better, it is essential to consider the range of challenges impacting both organizers and participants seeking for unforgettable experiences.

Thus, maintaining high levels of engagement is essential for event organizers who want to create meaningful experiences and foster a sense of community among attendees. However, participants often lose interest due to difficulties in discovering relevant events or a lack of interactivity within platforms. Studies show that engagement and personalization are top priorities for event organizers, with nearly 65% indicating that creating immersive experiences is essential to retaining attendee interest. [8]

Personalized recommendations are another pressing need, as today’s users expect suggestions that reflect their unique interests. Despite this, many platforms have a lack in utilization of advanced algorithms and data analytics to provide customized experiences effectively, but nearly 72% of users of such systems prefer platforms that recommend events based on past preferences and interactions, indicating high demand for enhanced personalization in event discovery and engagement. [8]

The lack of integrated functionality - such as combined ticketing, registration, and promotion - further complicates the user experience. As a result, organizers rely on multiple systems, leading to inefficiencies and a fragmented user experience. According to industry data, over 70% of event professionals view streamlined integration of digital tools as crucial for improving operational efficiency and attendee satisfaction. [8]

Scalability is also crucial as events grow more complex. Platforms must be able to handle increased traffic, especially during peak registration periods. Limited scalability can lead to system

downtime, slow response times, and potential loss of attendees, particularly for large-scale events. In 2023, the global events industry was valued at $1,135.4 billion according to [7], with projections for continued growth, highlighting the need for platforms that can handle high traffic without compromising

Finally, with the increasing volume of sensitive user data collected, platforms must prioritize data protection to maintain user trust and comply with regulations like GDPR [25], as data security breaches can significantly damage platform reputation and user confidence but what more important owners can face a lawsuit from regulations of countries over the world what can bring like financial and reputation damage. According to research [9], over 60% of attendees consider data security a primary concern when registering for events online, making security measures a one of the critical factors for success of organization operation on event planning market.

These challenge both event organizers and participants. For organizers, limited integration and scalability issues increase operational costs and complexity, while the lack of robust engagement and personalization features limits user retention and satisfaction. For participants, frustrations with event discovery, limited interactivity, and data security concerns create barriers to a seamless and enjoyable experience. From the other side regarding recent market research indicates a strong demand for improvement in these areas. For instance, the events industry is expected to continue its rapid growth, driven by digital innovation and heightened user expectations. Projections [7] suggest that by 2032, the global market could reach nearly $2.1 trillion, underscoring the need for platforms that deliver integrated, engaging, and secure experiences for users.

1.2. Selection of criteria for comparison

As mentioned earlier, a meaningful comparison of event planning platforms requires specific criteria. For this analysis, with a focus on three main factors: popularity, range of features, and user base based on Eventbrite article [11] and basic knowledge and logic. These criteria were chosen due to their relevance in evaluating platform strengths and weaknesses. Popularity provides insights into market acceptance and reliability. The range of features highlights a platform’s versatility, encompassing tools like attendee tracking, payment processing, and event website creation. The user base reflects suitability for different event types, demonstrating flexibility in diverse user’s needs. This way selected criteria will allow us to see most of the advantages and disadvantages during the comparison among established market leaders, but let’s first of all looks at each of them individually.

“Eventbrite” is a popular platform designed to simplify event creation and management across a broad range of event types. Its main features include comprehensive tools for creating events, integrated ticketing options, and advanced search and discovery functionalities, which help users find

events aligned with their interests. “Eventbrite’s” business model is centred around ticket fees, which allows organizers to access its features without an upfront cost, making it appealing to both individuals and businesses. [11], [12]

“Meetup” primarily serves community-focused events and organizers seeking to foster social connections around shared interests. This platform’s core features focus on creating groups, managing RSVPs, and promoting social gatherings, emphasizing ease for casual, community-driven events. Unlike platforms focused on large-scale events, “Meetup” uses a freemium model, allowing free use with an option to subscribe for additional features, making it accessible to hobbyists, social clubs, and local groups. [13]

“Ticketmaster” is a highly recognized platform primarily serving large-scale, ticketed events, including concerts, sports games, and theatre performances. Its key features include robust ticketing capabilities, reliable handling of high-traffic volumes, and integration with event discovery channels, making it a go-to solution for organizers of large events. The business model is based on ticket fees, which aligns with the needs of major event organizers looking for a stable, high-capacity platform. [14]

1.3. Evaluation criteria and best practices of event planning industry

The analysis of competitor platforms reveals several best practices that are essential for success in the event planning industry. These practices, which include organized design of web interface, community engagement features, integration with third-party tools, and robust data security, represent core standards that event platforms must meet to remain competitive.

Also, based on the analysis well-designed, intuitive interface is vital for engaging users based on one of the lead platforms articles [11], as ease of navigation significantly enhances the user experience. Leading event platforms emphasize simplicity, which allows users to easily discover, plan, and manage events without unnecessary complications.

Community engagement also plays a crucial role in maintaining user retention. Platforms that foster a sense of community by enabling users to form groups based on shared interests encourage long-term engagement and create a more socially dynamic environment.

Integration with third-party services, such as payment systems, social media platforms, and analytics tools, is essential for enhancing platform functionality and versatility [9]. This capability allows event organizers to manage payments, promote events, and monitor engagement seamlessly.

Data security and privacy are critical components in establishing user trust, especially in light of increasing privacy concerns. Platforms that prioritize strong data protection practices, such as

GDPR compliance, data encryption, and secure authentication methods, set a high standard for reliability and trustworthiness in the industry. [25]

1.4. Gaps in existing solutions

However, analysing existing platforms also reveals several problems that prevent them from fully satisfying user needs. [11], [12], [13], [14]

Firstly, many platforms do not have advanced recommendation algorithms [17], providing only basic search functions. This limitation reduces the level of personalization making it difficult for users to find activities that match their interests. In addition, platform specialization limits flexibility. For example, “Meetup” is primarily focused on community meetings while “Ticketmaster” is focused on large events and ticket sales. This narrow specialization limits the capabilities of companies and organizers who need a universal platform to support events of different types and sizes.

Another challenge is mobile usability, as some platforms offer users mobile apps with limited functionality and performance issues during use, leading to serious problems compared to the web version. But it is worth noting that not all the analysed apps are well optimized for web as well, and by not being better optimized, provide an inconsistent user experience, which can lead to the loss of potential customers.

Next problem is a limited customization available to business clients what as a result can hinder brand engagement, as most platforms lack the tools needed to tailor event pages or access detailed analytics on attendee behaviour, which as mentioned is not a good thing especially for the customers who a care about advertisement possibilities. [8]

Scalability also poses a challenge. While platforms like “Ticketmaster” are built for high-traffic events, others struggle to handle large, as even during the analyses was discovered constantly freezing and lagging of “Eventbrite”, “Meetup”. As events grow, these scalability limitations can lead to slowdowns and diminished user experience, what will affect the business not from the good side. From this we can draw a small conclusion that for many users attending events from mobile devices, seamless user interfaces, and scalability and optimization of the system are a must for a well build event planning and management application. [7]

1.5. Conclusion on analysis

Based on the analysis of existing platforms, there are clear opportunities to develop a competitive, user centred event management platform by addressing identified gaps and integrating industry best practices.

The primary market challenges - maintaining user engagement, delivering personalized experiences, integrating multiple functionalities, ensuring scalability, and meeting strict data security standards- reveal specific areas for improvement. A successful platform should include advanced personalization through recommendation algorithms to better match users' interests, enhancing engagement and retention. Additionally, integrating core functions like ticketing, registration, and promotions into a seamless interface could streamline event management for organizers and participants alike.

Furthermore, optimizing mobile usability and scalability is crucial. This includes addressing performance issues that occur in current platforms under high traffic, especially in case of peak usage times. Customization and data analytics for business clients represent another essential feature, allowing companies to brand their event pages and gain insights into attendee behaviour, thus enhancing promotional capabilities and user satisfaction. Given the rising emphasis on data protection, robust security features are critical to build user trust and comply with regulations like GDPR. [25]

Thus, it can be concluded that this analysis set a foundation for creating a valuable event management solution that not only meets industry demands but set foundation for developing a platform prototype that integrates these insights, aiming to provide a scalable, adaptable, and user-oriented solution.

2. TECHNICAL TASK

APPROVED

Supervisor: Assoc.Prof.Dr.Ilona Veitaitė

Bachelor student: Dzmitry Papkou

Date: 2024-09-09

1. **TITLE OF THE WORK (TITLE):**

   "Event Goose": Event Organisation and Registration Platform

2. **CONTENT OF THE ANALYTICAL AND RESEARCH WORK:**

   2.1. Conduct analysis of the market to define existing event planning solutions to understand market trends and identify current gaps in these solutions;

   2.2. Conduct comparative analysis of "Event Goose" with existing solutions to highlight unique features and competitive advantages.

3. **DESIGN SYSTEM FUNCTIONS:**

   3.1. Allow users to input event details such as location, dates, and optional tags or categories to create and manage events within specific regions;

   3.2. Allow non-registered users to search and view available events, with the ability to register a user to proceed with registration selected event;

   3.3. Provide functionality for users to read and post content related to their participation in offered events (post functionality only for registered users);

   3.4. Integrate features that will recommend events based on user preferences and defined financial limits (only for registered users);

   3.5. Enable registering for events directly through the platform (only for registered users);

   3.6. Implements basic platform security (like authorization and authentication, secure password storage etc.);

   3.7. Registered users (without paid subscription) would be limited to get discount or sales offers on events.

   3.8. Offer user subscription model where users pay for premium features, such as maintaining an unlimited history of past and future events planning, ad-free account and early access to new features at a platform;

   3.9. Offer businesses to have separate subscription plan to list their events' advertisement at platform, enhancing events visibility and promotion to potential customers.

4. **SYSTEM DESCRIPTION DOCUMENTATION AND INSTRUCTIONS:**

4.1. Terms of Service and System Policy for the user;

4.2. System management, content moderation, and user support documentation for the administration.

5. **SYSTEM DESIGN TOOLS, SOFTWARE AND HARDWARE REQUIREMENTS:**

5.1. System design: draw.io;

5.2. Development environment - VS Code text redactor;

5.3. System development tools: Java 21 (Spring Framework Environment) for backend, TypeScript (React library), CSS for frontend, and Docker for containerization;

5.4. Database management systems: PostgreSQL and S3 bucket for file storage;

5.5. Testing suite: Jest, JUnit, Postman, Firefox Developer Tools;

5.6. Version control system tool – Git/GitHub/GitLab;

5.7. Cloud service - AWS for hosting and managing system;

5.8. Operating system environment - Linux based (Ubuntu/Debian);

5.9. Hardware requirements: x86-64 processor with at least 4 cores, at least 4GB RAM, at least 20GB of storage;

5.10. APIs of third-party services: location and map services, event booking, payment processing, AWS tools;

5.11. Payment gateways: Stripe and/or PayPal.

6. **SYSTEM TESTING AND EVALUATION:**

6.1. Preparation of system testing strategy;

6.2. Evaluation of test results;

6.3. Comparison of the designed system with functional and business requirements.

7. **THESIS PRESENTATION REQUIREMENTS:**

7.1. The thesis description completed in accordance to the methodological requirements for the final thesis;

7.2. Presentation of the thesis include multimedia elements such as screenshots, diagrams, and live demo;

7.3. The defence of the thesis should last 6-8 minutes, including an oral presentation and a slide presentation.

3. PROJECT DESIGN

The main purpose of the “Event Goose” project is to optimize and simplify the events planning process on the most favorable conditions for individuals and organizations by integrating into a single platform for management such aspects of event management as search for available events, managing participant registrations, and coordinating event details. This is important because plan activities often involve switching between multiple services and platforms, which can complicate the process of organizing and managing your events. Also, many find it difficult to align their desires with market offers on a single platform, which can lead to a negative influence on the user experience.

The implementation of “Event Goose” is planned as a web-based system, which will ensure its multi-platform and ease of management. The backend of the system will be developed using Spring Framework and the frontend will be created using React library, which provides a dynamic user interface. With the help of this configuration, it will be possible to achieve one more purpose of the information system as accessibility for users on different devices and operating systems, which in turn will possibly expand the number of potential customers.

3.1. General characteristics of the researched problem area

Events planning is not an easy task, at least it includes setting up the event details, managing participant registrations, promoting the event, and ensuring a smooth execution in order to get the most positive experience from the future activities. However, it is because of the variety of actions that need to be taken, individuals and organizers face the problem of integrating all aspects efficiently, which can lead to oversights, poor attendance, or a mismatch between the promised event’s-based experiences and the actual outcome. “Event Goose” is a solution that allows to streamline this multifaceted sphere through a platform web interface, solving specific problems such as integration of disparate event management services and their alignment with promoting events to the right audience, and streamlining participant registration.

3.2. Information management policy of “Event Goose”

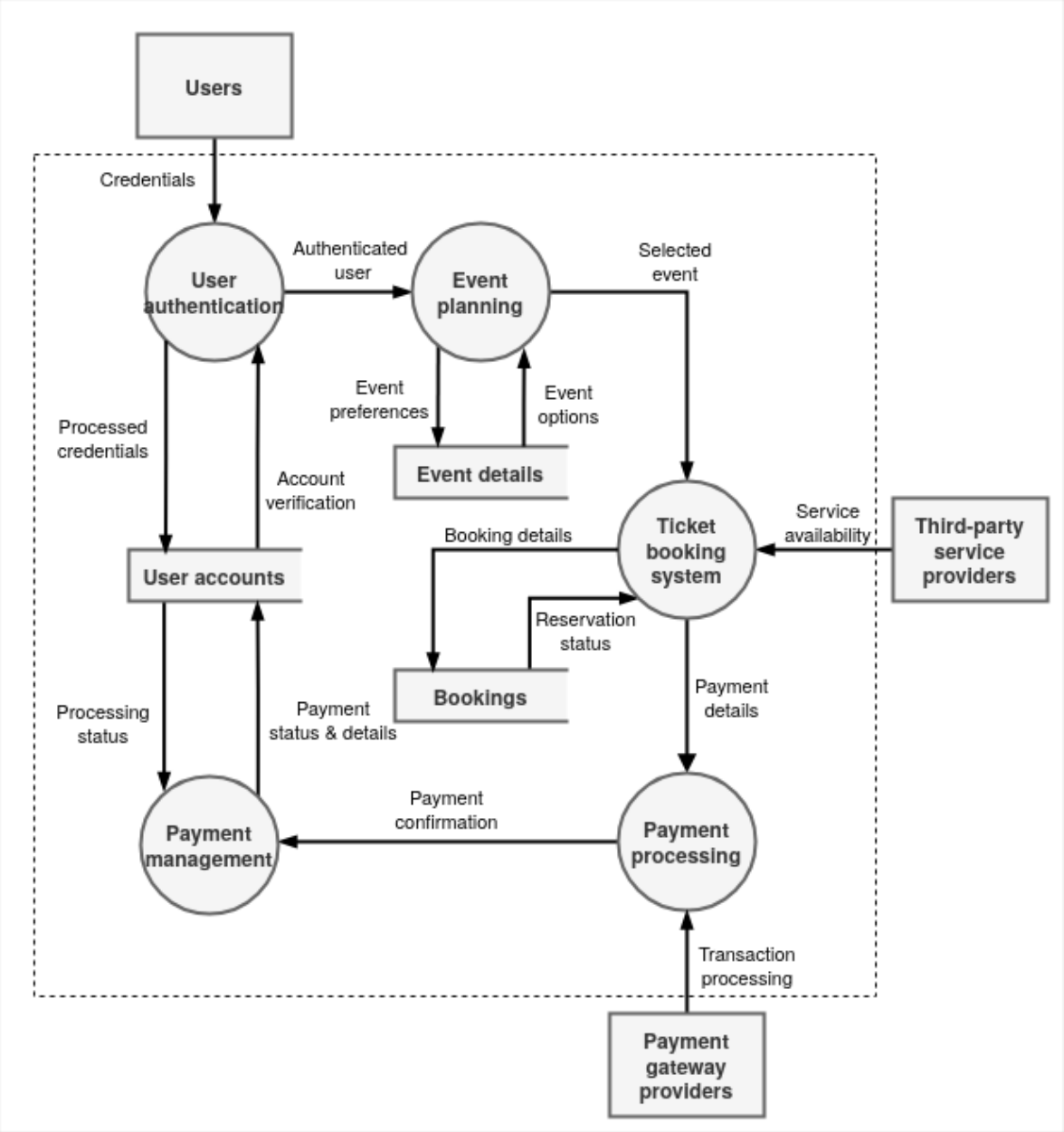
“Event Goose” follows a robust information management policy that ensures data integrity, confidentiality, and availability. The policy includes procedures for collecting, storing, processing and sharing of only necessary for work of the system data. Special attention is paid to compliance with global data protection regulations such as GDPR [25], encrypt the sensitive information is encrypted without compromising application performance. Also, the current policy giving users

options/control over their information by allowing to delete, edit, and request only the information that is necessary and related directly to the user making the request.

3.3. Information flow analysis

The efficient orchestration of data is very important for providing a good user experience. In this subchapter, will be provided overview of information flow within “Event Goose”, explaining data moves and transform across the system using a Data Flow Diagram.

The diagram below (see Figure 1) shows the user journey from authentication to payment, emphasizing the seamless integration of services provided by “Event Goose”.



Source: created by author by Dzmitry Papkou (2024) based on Yourdon and DeMarco notation.

Figure 1: **Data Flow Diagram for “Event Goose” information flow**

Describing a whole data flow process, where everything starts from users are authenticated when they begin interacting with the system. The system queries user preferences, which are then passed to the event planning module. This module interacts with the database to obtain personalized event recommendations based on the user's interests and past interactions.

At the same time, third-party service providers are integrated into the architecture to provide real-time availability and booking capabilities if selected event provided by platform partners.

Once services are selected, the reservation system connects the user's chosen options and sends them to confirm the booking. The payment process then begins, where transaction details are processed and handled, calling the payment gateway providers to complete the financial transactions if such exists.

3.4. Information flow analysis

One of the most basic things about any system is the functions it performs, because without a clear understanding of what the system is supposed to do, it is difficult to design anything. In this subchapter, will present the core functional and non-functional requirements for "Event Goose" based on previously conducted event planning industry research in the chapter 1 in order from most important to least important to give a deeper understanding planned system design concept and its functional purpose.

Functional requirements:

1. Search and filter events options based on user preferences;
2. Generate events recommendations based on user actions at the platform;
3. Integrate with third-party services for gathering more events;
4. Allow users to customize and save their event plans;
5. Allow users to create accounts and manage its settings;
6. Authenticate and authorize users to access different parts of the platform based on their user roles and permissions;
7. Support multiple languages;
8. Provide customer support via email;
9. Allow users to share information about the events.

Non-Functional requirements:

1. System uptime shall be 98% (2% for technical break and unexpected situations);

2. The response time for any action shall not exceed 20 seconds;
3. The system must support at least 1000 concurrent users;
4. Compliance with GDPR and other relevant data protection regulations;
5. The system must have scalability to accommodate a growing data volume.

Following given requirements, the Use Case diagram below (see Figure 2) offers a visual representation of the system's functionality from the perspectives of different actors within the “Event Goose”.

Source: created by author by Dzmitry Papkou (2024) based on UML standards notation.

Figure 2: **Use Case Diagram for “Event Goose” system**

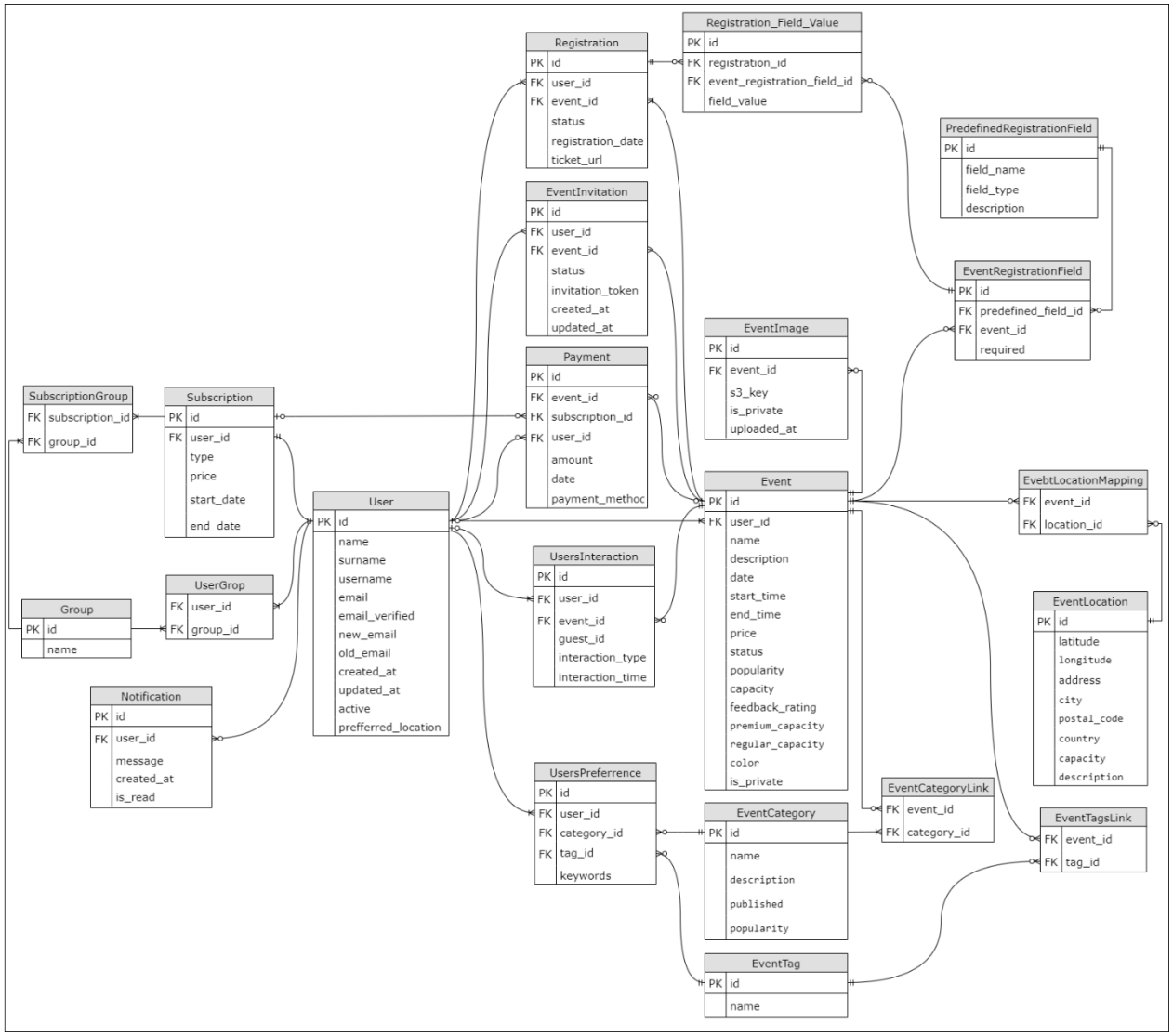
Taking a closer look at the diagram, we can identify three main actors and fifteen use cases for those users.

Where:

- Registered users are empowered to plan events, manage event registrations, access premium features (if subscribed), and engage with community-driven content by sharing their event experiences.
- Guest users can view event details and register new accounts.
- Administrators handle content management, monitor system health and security, and maintain user accounts.

3.5. Selection and description of computerized tasks

In the event planning industry, “Event Goose” implements many computerized tasks to ensure the most positive user experience and the smoothest system operation. The diagram below (see Figure 3) illustrates the relationships between the different entities within the system.



Source: created by author by Dzmitry Papkou (2024) based on Chen notation

Figure 3: **Entity-Relationship Diagram for “Event Goose” database schema**

Here is a breakdown of the tasks presented in the diagram:

- **User**: The system manages user information through the User entity, handling attributes such as email, username, roles, and subscription status.
- **Event Planning**: Users can plan or attend events, which are represented by the Event entity. This includes attributes such as event name, description, date, location, and ticket price. Event are categorized through the EventCategory entity, and specific details of the event location are managed through the EventLocation entity.
- **Event Registration**: Users can register for events through the Registration entity, which tracks the event, user, and registration status. Additionally, users can invite others to events, tracked by the EventInvitation entity.
- **Payment Processing**: All financial transactions are managed through the Payment entity. This includes payments related to event registrations, subscription fees, and any other financial interactions within the system.
- **Event Comments**: Users can get feedback thought notification system or ask questions about events, which are managed by the Notification entity, allowing interaction and engagement with other users.
- **Subscription Management**: Users may have different access levels based on their subscriptions. The Subscription entity tracks the user's subscription type and associated features, while the SubscriptionGroup and Group entities manage the grouping of users based on their subscription tiers.
- **Location and Region Management**: Events are tied to destinations and regions based on the country, managed by the EventLocation entity.

3.6. Description of computerization tools

The development of "Event Goose" will utilize a range of computerization tools, starting from using draw.io for the system and analysis diagrams, as this tool is well known for easy for utilization and is free to use. For the backend will be used Java 21 with Spring Framework Environment based on the Gradle build tool. The frontend will be build using React library with TypeScript shell to create more reliable and dynamic code and CSS for design. Also, data storage is planned to use such technologies as PostgreSQL for general data, and S3 for files storage. Both frontend and backend technologies like Jest and JUnit and external tools like Postman, Firefox Developer Tools will be suited for testing the application. Speaking about the deployment of the system, it will include using Docker for containerization and AWS as cloud hosting for the application and for CI/CD will be

utilising Git version control system tool with code upload to GitLab with automatic duplication to private GitHub repository as reserve backup copy.

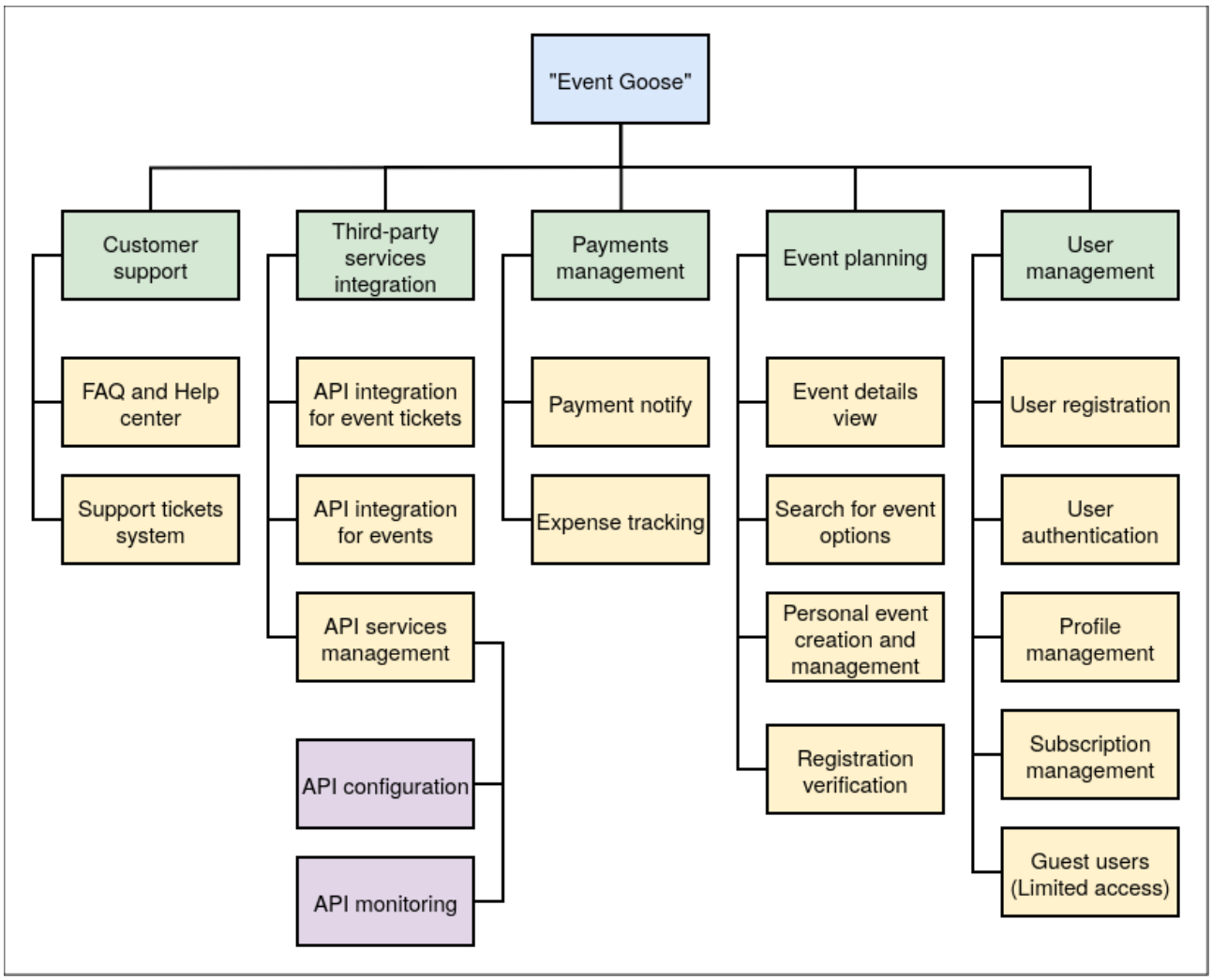
3.7. Business model

Talking about the planned business model for “Event Goose” can have several vectors of development.

- **Freemium:** Firstly, the simplest and the most user-oriented way of business development is the “freemium” model when basic features will be available to all users for free, but if a user wants an extended version with unlimited history of past and future trips, ad-free account and early access to new features on a platform, he will be charged a small subscription fee in line with market prices.
- **Business subscription:** In addition, it is planned to introduce a separate subscription/payment model for businesses that can advertise and promote their events or services on the “Event Goose” platform.
- **Referral commissions:** The last vector of the planned business model is to receive referral commissions from external booking sites like www.eventbookings.com using affiliate programs.

3.8. Hierarchy of computerized function

The hierarchical structure of the “Event Goose” is designed to organize and streamline the various computerized functions of the developed system, where clearly provided five main computerized functions: customer support, third-party services integration, payments management, event planning, and user management.



Source: created by author by Dzmitry Papkou (2024) based on the Functional Hierarchy Diagram notation

Figure 4: **Hierarchy of Computerized Functions Diagram**

Based on the updated diagram (see Figure 4), we can see how the functions are organized within the event planning system. Below is a detailed description of each responsibility:

Customer Support:

- FAQ and Help center: Provides users with answers to common questions and troubleshooting guides to resolve issues quickly.
- Support tickets system: Allows users to submit and track support tickets for personalized assistance and issue resolution.

Third-party services integration:

- API integration for event tickets: Connects with external services to manage event ticketing and bookings seamlessly.
- API integration for events: Links with external event services for broader event booking and management options.

- API services management:
    - API configuration: Manages the setup and configuration of APIs to ensure proper integration with third-party services.
    - API monitoring: Monitors the performance and uptime of APIs to ensure a smooth user experience and quick issue detection.

Payments management:

- Payment notify: Provide user notifications for event-related expenses.
- Expense tracking: Tracks all expenses associated with events, including ticket purchases, venue costs, and other related fees.

Event planning:

- Event details view: Allows users to view detailed information about events, including date, location, and ticket availability.
- Search for event options: Provides users with search functionality to discover and filter events based on preferences such as location, category, and price.
- Personal event creation and management: Users can create and manage their own events, handling details such as event description, date, and ticket pricing.

User management:

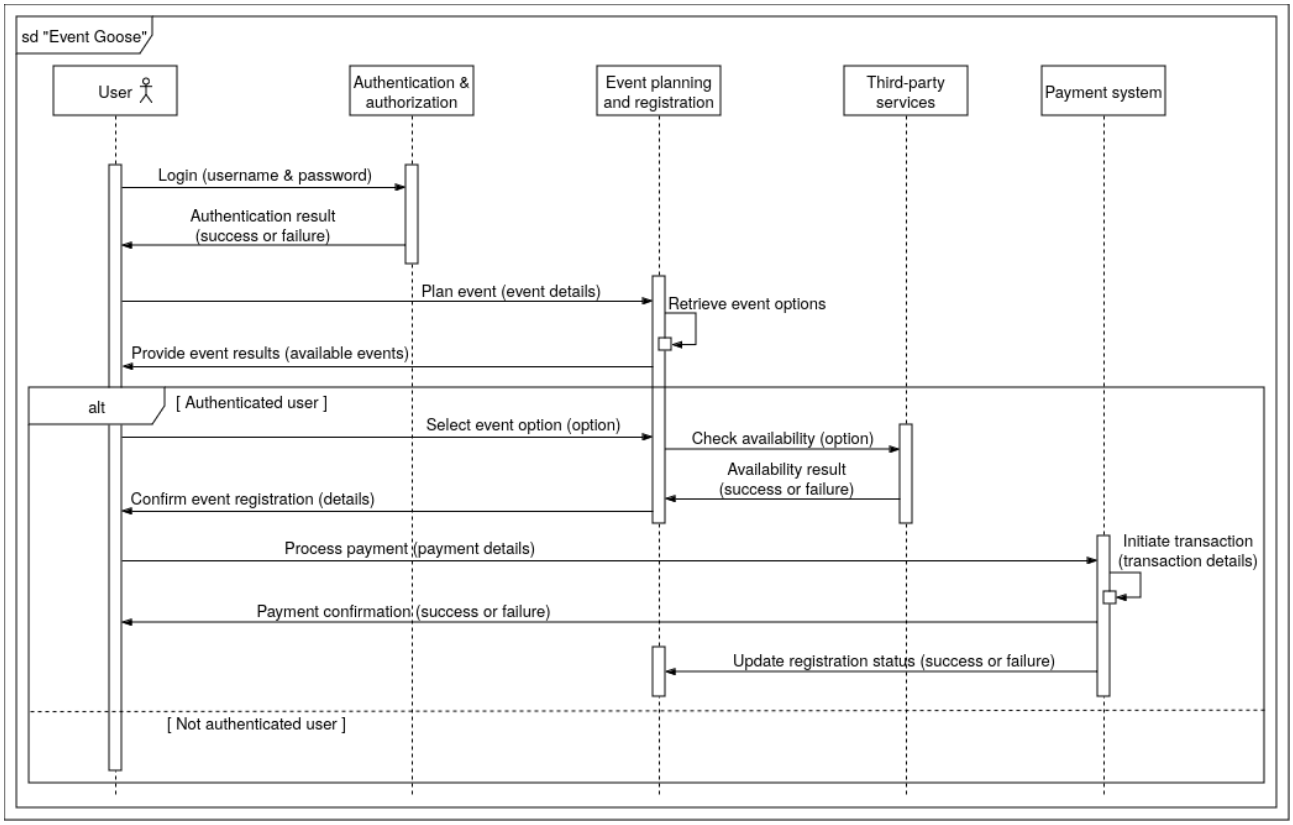
- User registration: Facilitates the creation of new user accounts.
- User authentication: Manages user login and authentication to ensure secure access to the platform.
- Profile management: Allows users to manage their personal profiles, including preferences, event history, and settings.
- Subscription management: Handles user subscriptions, including billing and access to premium features.
- Guest users (Limited access): Provides limited functionality for guest users, allowing them to view events and explore features without registering.

3.9. Computerized system data flow

In this subsection, will be explained in details the data flow in the “Event Goose” system using a sequence diagram below (see Figure 5). This diagram illustrates the user journey from authentication to payment, emphasizing the integration of the services provided.

It all starts with users logging in using their username and password. The “Authentication and authorization” component processes these credentials and returns the authentication result. For

authenticated users, the system allows them to plan events by retrieving event data and options from third-party services. Once the user selects an event, the system checks availability, processes the payment for event registration, confirms the payment, and updates the registration status in the "Event Goose" system. Unauthenticated users are restricted in their actions and can only view event options but cannot proceed with registration, payments, or event management.



Source: created by author by Dzmitry Papkou (2024) based on UML standards notation

Figure 5: **System Sequence Diagram**

3.10. Conceptual object model

The conceptual object model for this project focuses on the central entity: Event, with all other entities revolving around the management and organization of events, aligning closely with the entity relationship diagram described previously (see Figure 3).

Events table itself contains details such as the event's unique ID, name, description, date, time, and capacity. At the same time, each event is linked to a User through the *user_id* foreign key, representing the creator or manager of the event, whose data are stored in the Users table, with attributes like name, email, and username, as well as roles and permissions defined through associations with the Groups table. These associations are managed by the User Groups table, enabling role-based access control for event management.

Events are further enriched by their geographical details, with links to Event Locations, which describe the venue's name, address, and capacity. The Event Locations table connects events to the destinations and countries they occur in, via foreign keys referencing the Destinations and Countries tables. Destinations and countries are defined hierarchically, with destinations linking to countries and potentially multiple Regions for more granular location data.

Users can explore and engage with events through various categorizations. The Event Categories table and the Event Categories Links table define and assign categories to events, enabling users to search for and filter events based on their interests.

Event participation is tracked through the Registrations table, which stores user registrations for events, where each registration is linked to both the event and the user, ensuring proper management of attendees and invitations for events are managed via the Event Invitations table, where users receive unique invitation tokens and interaction with events is further happens by the Comments table, allowing attendees to leave feedback or ask questions, linked both to events and users.

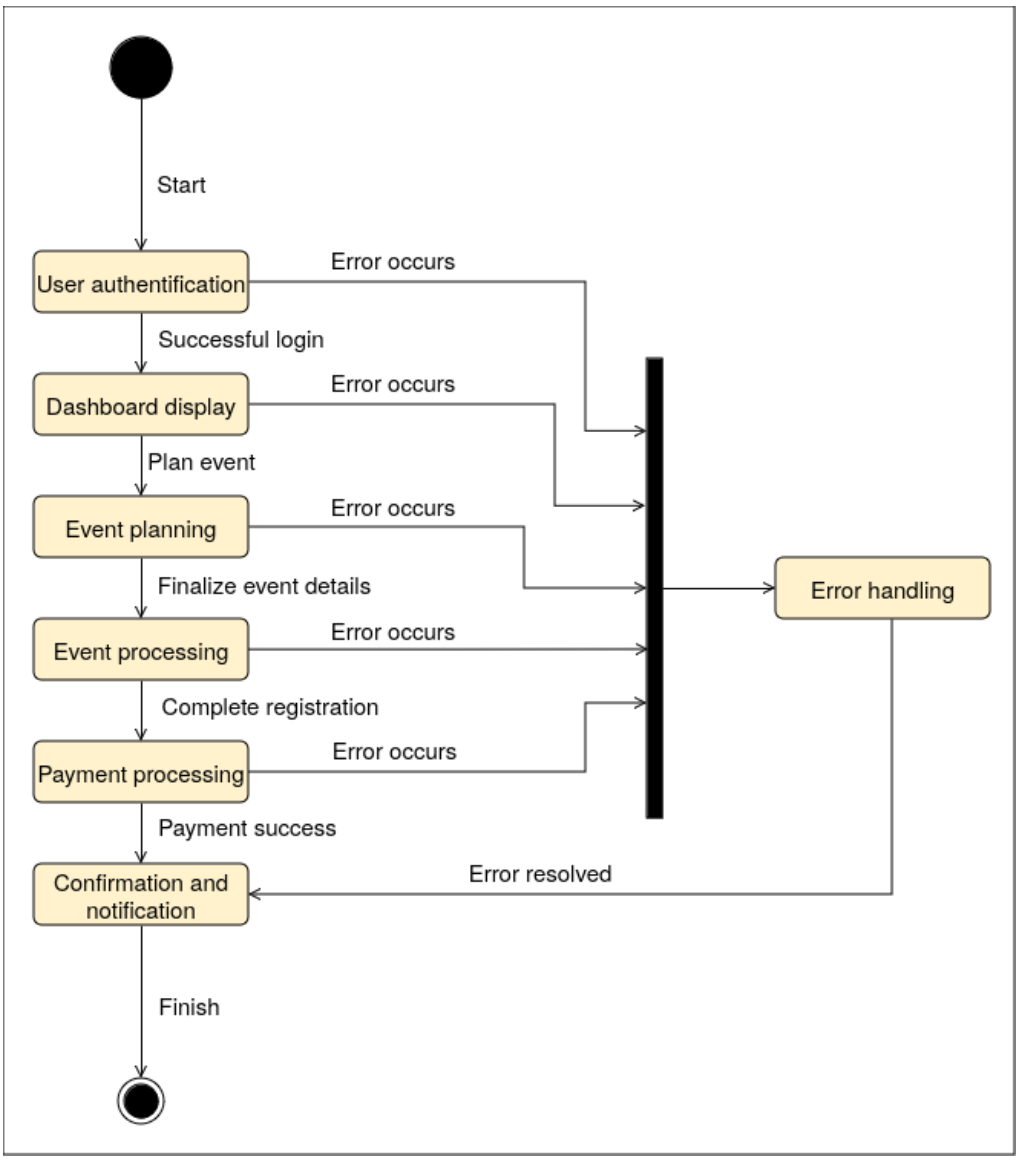
Payments for event participation and other related services are tracked in the Payments table. This table records transactions between users and the system for events or subscriptions, ensuring a comprehensive financial overview. Subscription services are managed in the Subscriptions table, with links to Groups through the Subscription Groups table, enabling premium features or access levels based on subscription status.

3.11. System states, processes, and functioning scenario description

This section shows the state of the art of the system designed to interact with users for event planning and registration purposes, guiding them through the entire process. The system is built to lead the user from authentication to final confirmation, including robust error handling implemented at each stage to ensure a smooth user experience.

The system's state is visually represented in the diagram below (see Figure 6), which illustrates the user's journey from the start of a session to the successful completion of a transaction, such as event registration. The sequence begins with the user logging into the system, where authentication is checked. If login issues occur, the system efficiently prompts the user to correct their credentials or restore their account if needed.

Once authenticated, the user proceeds to plan or create an event. At this stage, they can enter event details, review options, and finalize the event setup. The system incorporates error-catching mechanisms to detect and resolve data entry errors or system failures during this process



Source: created by author by Dzmitry Papkou (2024) based on UML standards notation

Figure 6: **System State Diagram**

. After finalizing the event details, the system handles event registration management and processes any related transactions. As in the previous steps, the system is equipped to manage issues such as payment failures or registration conflicts, promptly resolving them and keeping the user informed. The last stage involves confirmation and notification, where the system confirms the event or registration and notifies the user of a successful transaction, completing the main interaction between the user and the system.

3.12. A formal description of calculations

The core of the system is based on the capacity management of events. Each event has a predefined capacity C, which is stored in the Events table. The number of registered users R is

tracked dynamically through the Registrations table. Before allowing a new user to register for an event, the system checks if the available capacity is greater than zero:

$$C - R > 0 \quad (1)$$

If the event has reached its capacity (R =C), additional registrations are blocked. In certain scenarios, such as events offering priority registration for premium users, the total capacity can be divided into two segments:

$$Ctotal = Cregular + Cpremium \quad (2)$$

where Cregular is reserved for regular users and Cpremium is reserved for premium users. The system allows dynamic reallocation of unused premium spots to regular users as the event registration deadline approaches, using:

$$Cremaining = Cpremiumunused + Cregularunused \quad (3)$$

This ensures optimal usage of all available spots without overbooking.

In addition to capacity management, user preferences play massive role in event recommendations and registrations. The system should track user preferences in the Users table, including preferred event categories, locations, and pricing. When recommending events to a user, the system calculates a preference match score P:

$$P = \frac{M}{T} \quad (4)$$

where M is the number of attributes (such as category and location) that match the user's preferences, and T is the total number of relevant event attributes. If P = 1, the event fully aligns with the user's preferences. Events with higher P values are prioritized in the user's recommendations. To further enhance user experience, the system should evaluate the quality of each event using feedback from previous participants and event popularity. The event quality score Q is calculated as follows:

$$Q = \alpha \cdot F + \beta \cdot P + \gamma \cdot U \quad (5)$$

Where:

- F is the average user rating for the event.
- P is the ratio of registered users to total capacity:

$$P = \frac{F}{C} \quad (6)$$

- U fit is the preference match score calculated earlier.
- α, β, and γ are weight coefficients assigned to feedback, popularity, and user fit, respectively.

3.13. Comparison of similar software packages

Referring to the analysis section, this comparison examines the main features, strengths, and limitations of platforms like “Eventbrite”, “Meetup”, “Ticketmaster”, and the conceptualized “Event Goose”. By assessing how these platforms address user needs, personalization, flexibility, mobile usability, and business-specific customization, we can better understand the current market and identify ways to innovate within the industry. This section I will compare “Event Goos” to similar solutions, including “Eventbrite”, “Meetup”, and “Ticketmaster”, to evaluate how its features align with industry standards and where it may introduce unique value.

Table 1

Comparative Analysis of Events Management Platforms

Feature/System	**“Eventbrite”**	**“Meetup”**	“**Ticketmaster**”
Event creation and management	Comprehensive tools for creating and managing events	Suitable for community events but limited customization	Primarily for large events, concerts, and sports
Ticket sales integration	Integrated ticket sales and payment processing	Limited to RSVPs and external ticketing	Strong integration with ticket sales
User account creation	Yes	Yes	Yes
Event discovery and search	Advanced filtering and recommendations	Location and interest-based groups	Recommended events and shows
Multilingual support	Yes	Yes	Yes
Mobile application	Yes	Yes	Yes
Subscription model	Primarily free and event-based fees	Primarily free, focuses on social groups	Ticket fees are the primary revenue
Referral system	No	No	No
Data security	GDPR compliant, with encrypted transactions	Basic compliance, but limited features	High standards for large events
Scalability and reliability	Supports large events with high traffic	Suitable for small to medium-sized events	Designed for massive audiences

Source: created by author by Dzmitry Papkou (2024) based on comparative analysis of “Eventbrite”, “Meetup”, and “Ticketmaster” platforms

Based on the analysis table above (see Table 1), it is clear that “Event Goose” in theory meets most of the industry standards, offering to the user a wide range of functions not inferior to competitors can become a valuable platform in event planning industry.

Talking in the details we can see that table outlines the comparative analysis, examining core features across platforms such as event creation, ticket integration, user account functionality, and multilingual support. “Eventbrite” have a comprehensive event management tools, integrated ticketing, and advanced discovery functions, appealing to organizers of all sizes. “Meetup”, on the other hand, focuses on community-driven gatherings, with features centred on group creation and

social connection; it uses a freemium model to make basic services accessible while offering premium options for additional tools. “Ticketmaster” stands out for its strong ticketing capabilities and ability to manage large events reliably, catering to major entertainment venues and large-scale organizers through its established infrastructure and high-reliability system.

In contrast, “Event Goose” is envisioned to combine these strengths with several advantages. So, it should be designed to excel in user engagement through personalized recommendations based on user behaviour and preferences - an area where existing platforms offer limited support. Additionally, “Event Goose” could support a broader range of event types, allowing customization for both intimate gatherings and large-scale events, thereby providing flexibility that accommodates diverse needs. The inclusion of a referral system and a freemium subscription model introduces additional revenue streams, appealing to individual users and business clients alike. Prioritizing scalability and data privacy allow to address industry concerns over system reliability and user data protection. That way these differentiators will guide the system's development and design, ensuring that it meets both current industry standards and emerging user expectations

3.14. General guidelines for systems’ users and administrators

This subchapter provides general rules and guidelines for users and administrators interacting with the Event Goose platform. As the planned system is a web-based application designed to simplify the process of managing event-related aspects and registration at the events themselves, the platform is planned primarily as a service rather than a tool that does require the installation of additional software, so extensive guidelines and documentation are kept to a minimum.

3.14.1. Terms of Service and System Policies for the user

The following is a global set of rules and guidelines that define the expectations of all users, including individuals and organizations:

1. Access and platform usage:
    - Users are responsible for ensuring the accuracy of the information provided during account registration.
    - Event organizers must ensure that all event details, such as dates, descriptions, and pricing, are accurate and comply with applicable laws.
    - Users are prohibited from engaging in fraudulent or malicious activities, including creating fake events or misusing the platform to distribute inappropriate content.
2. Content guidelines:

- Event listings must not include offensive, harmful, or illegal content. All posted materials are subject to moderation.
- Users retain ownership of their content but grant "Event Goose" the right to use and display it for platform purposes.

3. Privacy and data processing:
   - The platform adheres to data protection regulations such as GDPR, ensuring the confidentiality and security of user data.
   - Users may request access to, edit, or delete their personal data in compliance with the platform's privacy policy.
4. System reliability and limits:
   - The platform is provided "as-is" and may experience occasional downtime for maintenance or updates.
   - Event Goose reserves the right to modify or discontinue features without prior notice.
5. Communication policy:
   - Users can reach out to the platform's support team through the provided contact email (support@traveldrago.com). Administrators will respond to user inquiries within 48 hours.

3.14.2. System management, content moderation, and user support guidelines for the administration

As for the administrator's guide, the currently planned implementation of the Event Goose demo does not provide a dedicated administrator interface, all administrative actions and support functionality are managed programmatically, but like the user, the administrator has responsibilities and rights, which are described below:

1. Account Management:
   - Administrators have the right to suspend the accounts of users who violate the Platform's Terms of Service.
   - Users will be notified of account suspension and may appeal the decision by contacting support.
2. Content Moderation:
   - Administrators are responsible for reviewing flagged content and ensuring that it complies with platform rules.
   - Decisions to remove content will be communicated to affected users within 48 hours.
3. Support and communication:

- Administrators handle all user queries, complaints and appeals through (support@traveldrago.com) email address.
- Responses are issued within 48 hours of receiving the inquiry.

4. Platform Maintenance:
   - Internal system logs and database queries are used to monitor the health and performance of the platform.
   - Administrators are responsible for implementing updates and resolving technical issues.
5. Security and Compliance:
   - All administrative actions are logged for auditing and reporting.
   - Administrators ensure that the system complies with relevant data protection regulations and report to address security vulnerabilities founded in the system.

3.15. Conclusion on project design

The project design of the "Event Goose" platform has laid the essential groundwork for creating a system aligned with the challenges and requirements of the modern event management market, provided in the analysis chapter. During this phase was analysed the core functionalities necessary for managing events and outlined a framework for implementing them within a time allocated for that project.

The most important component of the design phase was defining the hierarchical structure of data, which included users, events, categories, locations, and related entities. This was done by providing a database schema design and explained using an entity-relationship model that organizes data into interconnected tables. For example, events are linked to users, categories, and tags, providing a relational structure that supports advanced functionalities such as filtering and recommendations, while inclusion of tables like event locations, registration fields, and payments reflects the diverse aspects of event management, enabling the system to address both organizer and attendee requirements.

Moreover, the selection of modular system architecture dividing frontend, backend, deployment and hosting concepts promoting flexibility in for development, allowing different components to function independently while ensuring integration. The design also prioritized extensibility, ensuring that additional features or integrations can be seamlessly added in the future by leveraging a selection of established development stack, such as Java for the backend, React for the frontend and AWS for cloud hosting. This way, the specified system design provides a foundation phase sets the direction for the implementation, supporting the system's development adheres to the planned design and achieves its intended functionality.

4. INFORMATION SYSTEM PROJECT IMPLEMENTATION

This chapter will cover such aspects as the design and implementation of the system, the analysis of the information to be processed, and the specification of input and output data, as well as the classification of this information and the flow of data in the system in the form of data tables and illustrative examples of the information to be transferred through.

4.1. Description of the classification and coding system

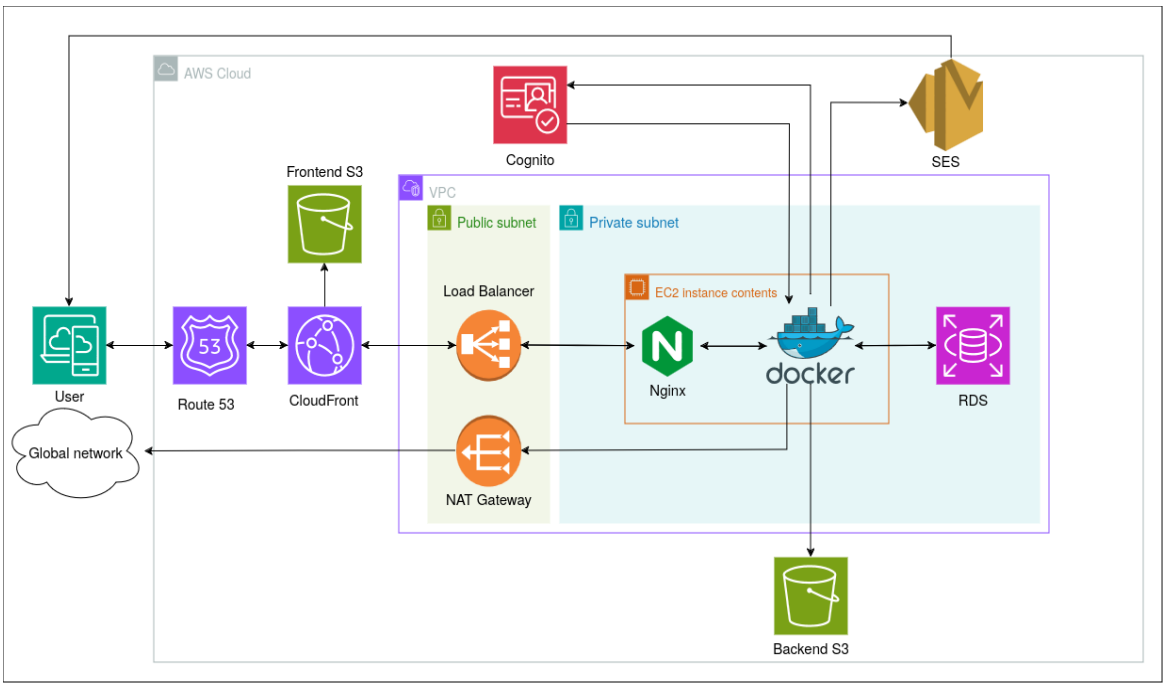
To provide users with a seamless event planning experience, the classification and coding system must enable efficient filtering and organization of event searches, primarily based on user preferences and secondarily according to the system's capabilities. The developed classification system, in its final state, should include key categories such as event types, locations, and user-specific preferences, allowing users to better tailor their event search and participation.

The hierarchical user authority structure within the system will ensure proper role-based access and control. Administrators will have full control over the platform, managing user accounts, handling event listings, monitoring platform usage, and resolving user issues. Event organizers will be responsible for managing and updating event details, handling registrations, and interacting with attendees. General users will use the platform to browse and participate in events, provide feedback, and interact with support for any platform-related requests.

The coding classification system for "Event Goose" will be designed to integrate its key components e.g. frontend, backend, and database while ensuring that each is thoroughly tested before integration. The frontend must be responsive, adhering to modern technological standards, and accessible across various devices, ensuring an intuitive and web interface. The backend will handle business logic, user authentication and authorization, and communication with third-party APIs for features like payments and notifications. The database will be structured to efficiently store and retrieve event data, supporting advanced query processing for searching, filtering, and categorization, while ensuring data integrity and security throughout the system.

4.2. System architecture design

The architecture of "Event Goose" system is built using AWS cloud services, with a focus on scalability, security, and availability, where the main components include a frontend hosted on AWS S3 through CloudFront, a backend system deployed on EC2 instances managed through Docker containers, and an integrated database via Amazon RDS.



Source: created by author by Dzmitry Papkou (2024) based on AWS Architecture Diagram Guidelines

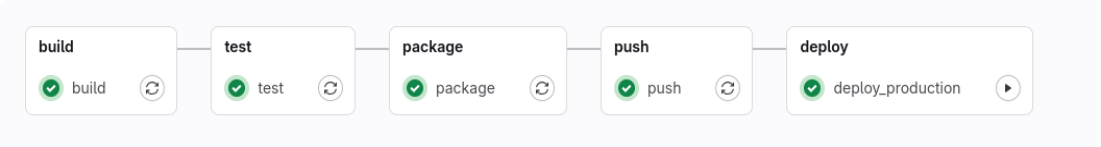
Figure 7: **AWS System Architecture**

The architecture diagram above (see Figure 7) outlines the key flows and components:

- Route 53: DNS service used to route incoming traffic to the appropriate cloud services.
- CloudFront: Content delivery network responsible for delivering static resources quickly to users across the globe, retrieving the static files from Frontend S3.
- Frontend S3: Stores static content like HTML, CSS, and JavaScript files that compose the user interface of the “Event Goose” application.
- VPC: The application infrastructure is hosted in an isolated network where security and access controls are defined. It includes both a public and a private subnet.
- Public Subnet: Hosts the Load Balancer and NAT Gateway, enabling incoming user requests to be routed to the private subnet, while allowing the backend components to access the internet securely.
- Private Subnet: Contains EC2 instances running Docker containers, which serve backend functionalities. The backend includes business logic, user management, and API communication.
- Nginx and Docker: The Nginx server running on EC2 acts as a reverse proxy to route traffic form Load Balancer to Docker containers which is used to manage backend application.
- RDS: This service utilized as the master database to store event-related data, user preferences, and a user itself.
- Backend S3: Used for storing different versions of backend configurations.

- Cognito: Handles user authentication and authorization, providing secure access control based on user roles, including admins, organizers, and general users, and hold user passwords for security reason.
- SES: Manages email notifications, providing the system with the ability to send updates and confirmations to users.

The *.gitlab-ci.yml* is a file used by frontend and backend with the simmilar staging structure but with different configurations for GitLab CI/CD pipelines to automate the build, testing, packaging, and deployment of the application.



Source: created by author by Dzmitry Papkou (2024).

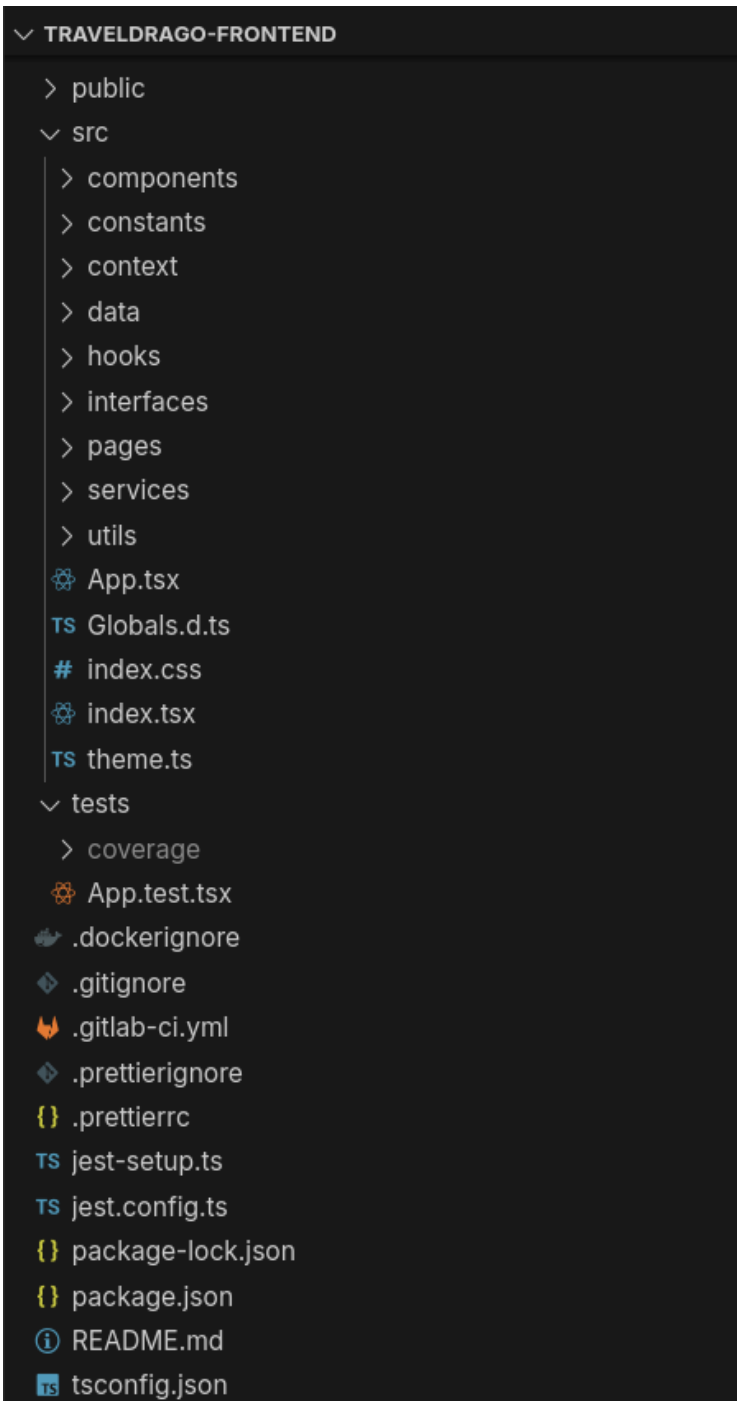
Figure 8: **GitLab CI/CD Pipeline**

The pipeline consists of the following stages of robust deployment process (see Figure 8):

1. Build: This stage compiles the codebase, ensuring that the system dependencies and components are properly built for the target environment.
2. Test: Unit and integration tests are executed during this stage to validate the correctness of the business logic, database connections, and third-party integrations.
3. Package: Once testing is successful, the application is packaged into archive, preparing it for deployment.
4. Push: The prepared archives are pushed to the AWS S3, making them available for the deployment phase.
5. Deploy: The final stage is confirmed manually by system maintainer and during this stage application is deployed to the production environment utilizing rollback mechanisms in case of unsuccessful deployment process.

4.3. Frontend architecture design

The frontend of the platform is organized as shown in the provided directory structure follows a modular approach where each functionality is grouped into relevant folders, ensuring clear separation of concerns.



Source: created by author by Dzmitry Papkou (2024).

Figure 9: **Frontend Package Structure**

As shown on the diagram above (see Figure 9) the public folder contains static assets that are directly served to the client, such as the index.html file and icons, while main application logic resides within the *src* folder, which houses different parts of the frontend system.

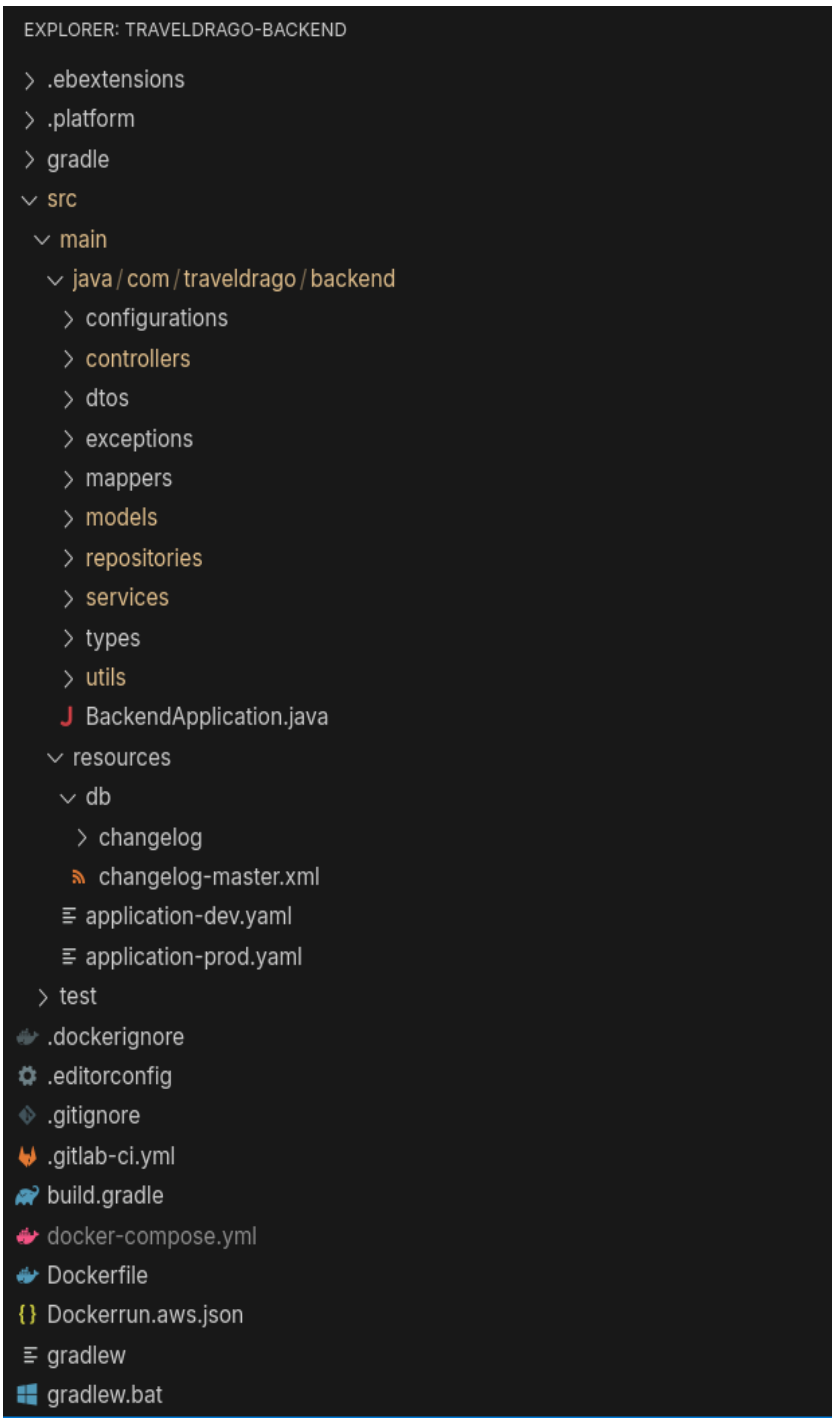
The *components* directory is responsible for holding reusable UI elements like buttons, forms, or modals, which are used across multiple pages of the application. Constant values used across the system, such as API endpoints or configuration variables, are stored in the *constants* folder. The

context directory is used for managing global state to share data across different components without prop drilling. The *data* folder contains mock data or data structures used for populating the frontend during development or testing phases. The *hooks* folder is designed to manage custom React hooks that encapsulate common logic, such as fetching data or handling user authentication, making them reusable across components. Interfaces, stored in the *interfaces* folder, define the TypeScript types and structures used across the project, ensuring strong typing and reducing runtime errors. The main entry point of the frontend application is the *App.tsx* file, where the application's layout and routing are handled. Global TypeScript types are stored in *Globals.d.ts*, and styling for the application is managed via the index.css and *theme.ts* files. The *index.tsx* is responsible for rendering the entire application to the DOM (Document Object Model).

Testing is handled through the test's directory, where unit and integration tests are written, ensuring that each component and service functions as expected. The *jest-setup.ts* and *jest.config.ts* files provide the configuration for the Jest testing framework. The gitlab-ci.yml file defines the continuous integration pipeline for the project, outlining the steps needed for building, testing, and deploying the frontend. Additionally, project dependencies are managed via the *package.json* and *package-lock.json* files, while Prettier configuration files handle code formatting.

4.4. Backend architecture design

The project backend uses a combination of Java, Gradle, Docker, and Liquibase for development, testing, and production deployment. The backend is designed with scalability and flexibility in mind, using Docker for containerization and Liquibase for managing database migrations.



Source: created by author by Dzmitry Papkou (2024).

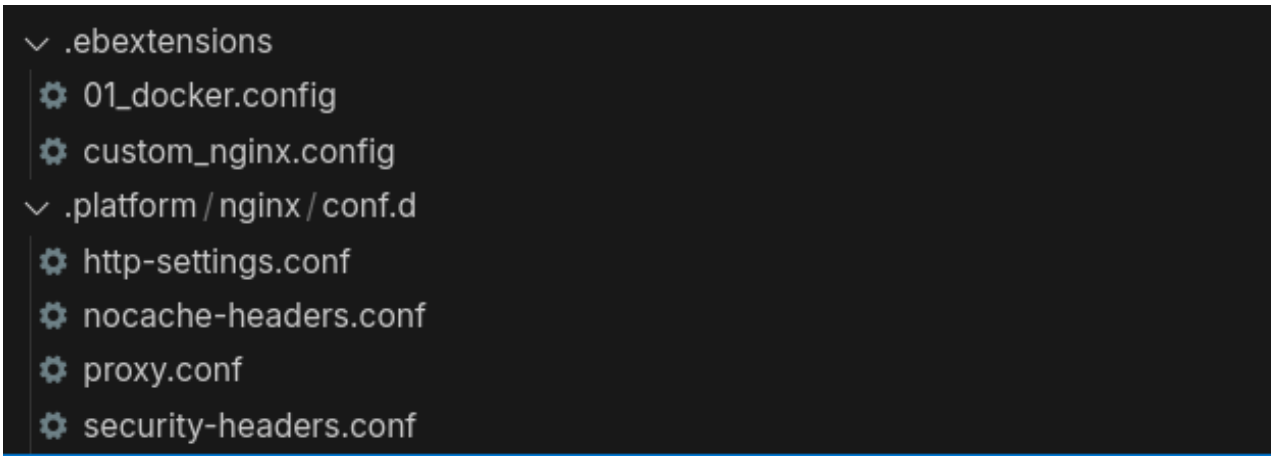
Figure 10: **Backend Package Structure**

As shown on the diagram above (see Figure 10) he *src* folder contains the core application logic, where the configurations directory purpose to carry out configuration-related code, such as web security settings, CORS policies, AWS connection configuration and low-level settings loaded during startup of the application to provide common component. Controllers in the *controllers* folder handle incoming HTTP requests, while *dtos* define the structure of data exchanged objects between the frontend and backend. The *exceptions* folder manages custom exception handling, ensuring consistent

error responses. Business logic resides in the services folder, and data is handled by *repositories*, which interact with the database using JPA repository dependencies. Models represent the application's core database entities, and mappers are used to convert between domain objects and DTOs.

Database management is achieved using Liquibase - a version control tool for database schema changes. In the *resources/db/changelog* folder, Liquibase configuration files such as *changelog-master.xml* define the database schema versioning for both development and production environments. The *application-dev.yaml* and *application-prod.yaml* files contain environment-specific properties, including database credentials, API configurations, and other settings required to run the application in different environments. This ensures that the backend behaves appropriately in both development and production setups.

For the deployment in the application is utilized *Docker*, that way the *Dockerfile* into the file structure specifying the steps required to build the backend into a Docker image. At the same time you can see a multi-container setup through *docker-compose.yml* file which is used to startup application and its dependencies, such as the database during local development. However, for production, the platform relies on stand-alone Docker containers defined in *Dockerrun.aws.json*, a configuration file that AWS uses for deploying backend on EC2 instance.



Source: created by author by Dzmitry Papkou (2024).

Figure 11: **Nginx and Docker Configuration for Backend**

The *.ebextensions* folder contains AWS Elastic Beanstalk configuration files that are very important for defining how the Docker containers should run in the AWS environment. The *01_docker.config* and *custom_nginx.config* files provide custom settings for the containerized application, including additional commnads to the Nginx configuration and Docker runtime options. The *.platform/nginx/conf.d/* directory includes several configuration files like *http-settings.conf*,

proxy.conf, and *security-headers.conf*, which customize how Nginx handles HTTPS requests, proxies traffic, and enforces security headers.

4.5. Input data specification

The input data specification verify that all necessary information is collected accurately for various user interactions within the event planning and management system. The following outlines the required fields for different functionalities in the system:

- **Login**: Users must provide their email or username and password to authenticate themselves within the system, what ensures secure access to personalized event management features.
- **Registration of a new user**: New users are required to input at least their email, password, and username to create an account. Additionally, users can specify an optional subscription status, which may grant access to premium event features or early registration privileges.
- **Event search**: Users input various event-related details such as event location (destination or country), event category (e.g., music, sports, conferences), event dates, and any specific preferences like event type or price range. This helps users find events that align with their interests and availability.
- **Event registration**: To register for an event, users must provide their user ID, event ID, and optional depending on the event the number of tickets they wish to purchase. The system in case of ticket specification calculates the total price based on the ticket quantity and ensures that the event's capacity is not exceeded before completing the registration.
- **Payment**: Users provide their user ID, payment amount, payment method, and payment date to complete transactions for event bookings or subscription services.
- **Event invitation management**: Users can input the event ID and the list of invitees (user IDs) when sending out event invitations.
- **Event ticket purchase**: When purchasing event tickets, users must input their user ID, event ID, ticket quantity, and ticket price.

4.6. Output data specification

The output data specification, based on the input functionality, is presented below in the form of REST API responses, typically formatted in JSON with expected outputs corresponding to each functionality and accompanied by appropriate REST response codes for both successful and failed operations.

Login:

Upon successful login, users receive the following data:

- *user_id*: The unique identifier for the user.
- *username*: The user's username.
- *email*: The user's email address.
- *role_id*: The user's role within the system (e.g., admin, regular user).
- *auth_token*: A token for session management and authentication.

Response codes:

- Success: 200 (OK) – User successfully authenticated.
- Failure: 401 (Unauthorized) – Invalid credentials.

New user registration:

After successfully registering a new user, the following information is returned:

- *user_id*: The newly created user's unique identifier.
- *username*: The username chosen during registration.
- *email*: The user's registered email address.
- *subscription_status*: The user's subscription status (if applicable).

Response codes:

- Success: 201 (Created) – User account successfully created.
- Failure: 400 (Bad Request) – Registration failed due to missing or invalid input.

Event search:

Users receive a list of events matching their search criteria, including:

- *event_id*: Unique identifier for each event.
- *name*: The name of the event.
- *date*: The date of the event.
- *location*: The location of the event (destination or country).
- *category*: The category of the event (e.g., music, sports).
- *price*: The price of the event.

Response codes:

- Success: 200 (OK) – Events successfully retrieved.
- Failure: 404 (Not Found) – No events found matching the criteria.

Event registration:

Upon successful registration for an event, users receive a confirmation with:

- *booking_id*: The unique identifier for the event booking.
- *user_id*: The unique identifier of the user registering for the event.
- *event_id*: The unique identifier of the event.
- *number_of_tickets*: The number of tickets purchased.
- *total_cost*: The total cost of the tickets purchased.

Response codes:

- Success: 200 (OK) – Event registration completed.
- Failure: 400 (Bad Request) – Invalid registration request or event is full.

Payment:

After successfully processing a payment for an event, users receive:

- *payment_id*: The unique identifier for the payment transaction.
- *user_id*: The unique identifier of the user making the payment.
- *amount_paid*: The total amount paid.
- *payment_method*: The method of payment (e.g., credit card, PayPal).
- *payment_date*: The date of the payment.
- *status*: The payment status (e.g., successful, pending).

Response codes:

- Success: 200 (OK) – Payment successfully processed.
- Failure: 402 (Payment Required) – Payment could not be processed.

Event invitation management:

After sending out event invitations, users receive:

- *invitation_id*: The unique identifier for the invitation.
- *event_id*: The unique identifier of the event.
- *invitee_list*: A list of user IDs for the invitees.
- status: The status of the invitations (e.g., sent, pending).

Response codes:

- Success: 200 (OK) – Invitations successfully sent.
- Failure: 400 (Bad Request) – Failed to send invitations due to missing or invalid input.

Event ticket purchase:

Upon successfully purchasing tickets for an event, users receive:

- *ticket_id*: The unique identifier for the event ticket(s).
- *user_id*: The unique identifier of the purchaser.
- *event_id*: The unique identifier of the event.
- *number_of_tickets*: The number of tickets purchased.

Response codes:

- Success: 200 (OK) – Tickets successfully purchased.
- Failure: 400 (Bad Request) – Invalid purchase request or event is full.

4.7. Database project

Based on the Entity-Relationship Diagram (see Figure 3), the following tables are presented database project of developed system with more detailed specifications of database entities, all including columns of the tables, data types, descriptions, and constraints.

Table 2

Users

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each user (Cognito sub)
name	VARCHAR (50)	NULL POSSIBLE	First name of the user
surname	VARCHAR (50)	NULL POSSIBLE	Surname of the user
username	VARCHAR (50)	NOT NULL, UNIQUE	Unique username for the user
email	VARCHAR (50)	NOT NULL, UNIQUE	Unique email address for the user
email_verified	BOOLEAN	NOT NULL	Whether the user's email is verified
new_email	VARCHAR (100)	NOT NULL	Temporary new email for the user
old_email	VARCHAR (100)	NOT NULL	Previous email of the user
created_at	TIMESTAMP	NOT NULL	Account creation timestamp
updated_at	TIMESTAMP	NOT NULL	Last update timestamp
active	BOOLEAN	NOT NULL, DEFAULT TRUE	Whether the account is active
preferred_location	VARCHAR (255)	NULL POSSIBLE	User’s preferred location by default the location of user from the UI

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Users table is designed to store information about users of the system. Each user has a unique identifier (*id*), which is a UUID that serves as the primary key. Additional fields store personal information such as name, surname, username, and email, table also includes verification status

(*email_verified*) and logs important timestamps for account creation (*created_at*) and updates (*updated_at*). The active field, with a default value of TRUE, determines if a user's account is active.

Table 3

Group

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each group
name	VARCHAR (255)	NOT NULL, UNIQUE	Name of the group

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Group table stores information about different user groups within the system. The primary key (*id*) is an auto-incrementing serial integer, ensuring each group is uniquely identified. Each group also has a name, which must be unique across the system.

Table 4

User Group

Column	Type	Constraints	Description
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
group_id	INT	NOT NULL, FK	Reference to the group (groups.id)

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The User Group table represents a many-to-many relationship between users and groups. It consists of two foreign keys: *user_id*, referencing the Users table, and *group_id*, referencing the Group table.

Table 5

Event

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each event
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
name	VARCHAR (255)	NOT NULL	Name of the event
description	TEXT	NULL POSSIBLE	Description of the event
date	DATE	NOT NULL	Date of the event
start_time	TIME	NOT NULL	Event start time
end_time	TIME	NOT NULL	Event end time
price	DECIMAL (10,2)	NULL POSSIBLE	Price for the event
capacity	INT	NULL POSSIBLE	Capacity of the event
popularity	INT	NULL POSSIBLE	
status	VARCHAR (50)	NOT NULL	Status of the event

Table 5
(continued)

Column	Type	Constraints	Description
is_private	BOOLEAN	NOT NULL, DEFAULT TRUE	Whether the event is private
color	VARCHAR (7)	NULL POSSIBLE	Event color for UI representation
feedback_rating	DECIMAL (3, 2)	NULL POSSIBLE	Average feedback rating
premium_capacity	INT	NULL POSSIBLE	Reserved capacity for premium users
regular_capacity	INT	NULL POSSIBLE	Capacity for regular users

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event table stores information about system events. Each event has a unique identifier *(id*), which serves as the primary key. The *user_id* field is a foreign key referencing the *Users* table, indicating which user created the event. It also includes fields for name, description, event *date*, *time*, and optionally, *price* and *capacity*. The status field indicates the state of the event, while *is_private* (default TRUE) indicates whether the event is private.

Table 6

Event Location

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each event location
latitude	DECIMAL (10, 8)	NOT NULL	Latitude coordinate
longitude	DECIMAL (11, 8)	NOT NULL	Longitude coordinate
address	VARCHAR (255)	NOT NULL	Detailed address of the location
city	VARCHAR (100)	NULL POSSIBLE	Name of the city
postal_code	VARCHAR (20)	NULL POSSIBLE	Postal code of the location
country	VARCHAR (100)	NOT NULL	Name of the country
capacity	INT	NULL POSSIBLE	Capacity of the location
description	TEXT	NULL POSSIBLE	Additional details about the location

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Location table contains information about venues where events are held. Each record includes a unique identifier (*id*), geographical coordinates (*latitude*, *longitude*), and address details (*address*, *city*, *postal_code*, *country*). The table also provides optional fields for *capacity*, specifying the maximum number of attendees, and *description*, allowing for additional details about the venue.

Table 7

Event Location Mapping

Column	Type	Constraints	Description
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
location_id	INT	NOT NULL, FK	Reference to the location (event_location.id)

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Location Mapping table establishes a many-to-many relationship between events and categories. It contains two foreign keys: *event_id* (referencing the Events table) and *category_id* (referencing the Event Category table), with both fields forming a composite primary key.

Table 8

Event Category

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each event category
name	VARCHAR (255)	NOT NULL	Name of the category
description	TEXT	NULL POSSIBLE	Additional information about the category
published	BOOLEAN	NOT NULL, DEFAULT FALSE	Whether the category is published
popularity	INT	NULL POSSIBLE	Popularity score of the category

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Category table stores predefined categories for events. It has a primary key (*id*) and a name field. This table is used to categorize events, allowing users to filter or group events by category.

Table 9

Event Categories Link

Column	Type	Constraints	Description
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
category_id	INT	NOT NULL, FK	Reference to the category (event_categories.id)

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Categories Link table establishes a many-to-many relationship between events and categories. It contains two foreign keys: *event_id* (referencing the Event table) and *category_id* (referencing the Event Category table), with both fields forming a composite primary key.

Table 10

Event Tag

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each event tag
name	VARCHAR (255)	NOT NULL	Name of the tag

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Tag table stores predefined tag for events. It has a primary key (*id*) and a name field. This table is used to tagged events, allowing users to filter or group events by tag.

Table 11

Event Tags Link

Column	Type	Constraints	Description
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
tag_id	INT	NOT NULL, FK	Reference to the tag (event_tag.id)

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Tags Link table establishes a many-to-many relationship between events and tags. It contains two foreign keys: *event_id* (referencing the Event table) and *tag_id* (referencing the Event Tag table), with both fields forming a composite primary key.

Table 12

Event Invitation

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each invitation
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
invitation_token	UUID	NOT NULL, UNIQUE	Unique token for the invitation
status	VARCHAR (50)	NOT NULL	Status of the invitation
created_at	TIMESTAMP	NOT NULL	Timestamp of creation
updated_at	TIMESTAMP	NOT NULL	Last update timestamp

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Invitations table stores invitations sent to users for events. Each invitation has a unique identifier (*id*) and references both an event (*event_id*) and a user (*user_id*). The *invitation_token* is a unique token for tracking each invitation, and the status field tracks the status of the invitation (e.g., accepted, declined).

Table 13

Notification

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each notification
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
message	VARCHAR (255)	NOT NULL	Content of the notification
created_at	TIMESTAMP	NOT NULL	Notification creation timestamp
is_read	BOOLEAN	NOT NULL, DEFAULT FALSE	Whether the notification has been read

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Notification table tracks system-generated notifications sent to users, where each notification is uniquely identified by an *id* and linked to a user via the *user_id* field. The *message* field contains the content of the notification, while the *created_at* field logs when the notification was issued and *is_read* field indicates whether the notification has been viewed by the user.

Table 14

Registration

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each registration
event_id	UUID	NOT NULL, FK	Reference to the event (events2.id)
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
registration_date	TIMESTAMP	NOT NULL	Date of registration
status	VARCHAR (50)	NOT NULL	Status of the registration
ticket_url	VARCHAR (512)	NULL POSSIBLE	Reference to the user ticket in private s3 bucket

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Registration table logs user registrations for events. It has a unique identifier (*id*) as the primary key and two foreign keys: *event_id* (referencing the Events table) and *user_id* (referencing the Users table). This table stores when a user registered for an event, along with an optional status field and a link to the secure S3 bucket cloud storage where keeped ticket confirming user reqistration under *ticket_url.*

Table 15

Payment

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each payment
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
subscription_id	UUID	NOT NULL, FK	Reference to the subscription (subscriptions.id)
amount	DECIMAL (10,2)	NOT NULL	Amount of the payment
date	TIMESTAMP	NOT NULL	Payment date
payment_method	VARCHAR (50	NOT NULL	Payment method used

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Payment table tracks all payments made within the system, both for events and subscriptions tracking the amount paid, the payment date (*date*), and the payment method (*payment_method*). It has a primary key (id), and foreign keys *user_id*, *event_id*, and *subscription_id*, referencing the Users, Event, and Subscription tables respectively, where constraint ensures that at least one of *event_id* or *subscription_id* is valid for each payment or set default value cast to "00000000-0000-0000-0000-000000000000" respectively.

Table 16

Subscription

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each subscription
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
type	VARCHAR (50)	NOT NULL	Type of subscription
price	DECIMAL (10,2	NOT NULL	Price of the subscription
start_date	DATE	NOT NULL	Subscription start date
end_date	DATE	NOT NULL	Subscription end date

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Subscription table stores information about user subscriptions. Each subscription has a unique identifier (*id*) and a foreign key (*user_id*) linking it to the Users table. The table purpose is to track the type of subscription, its price, and the start and end dates and to manage recurring or one-time subscriptions to system services.

Table 17

Subscription Group

Column	Type	Constraints	Description
subscription_id	UUID	NOT NULL, FK	Reference to the subscription (subscriptions.id)
group_id	INT	NOT NULL, FK	Reference to the group (groups.id)

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Subscription Group table links user subscriptions to groups, enabling special access or privileges based on subscription status. It contains foreign keys *subscription_id* (referencing the Subscription table) and *group_id* (referencing the Group table), with both fields together forming the composite primary key.

Table 18

Predefined Registration Field

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each predefined field
field_name	VARCHAR (255)	NOT NULL	Name of the field
field_type	VARCHAR (50)	NOT NULL	Data type of the field
description	TEXT	NULL POSSIBLE	Additional details about the field

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Predefined Registration Field table stores reusable fields that event organizers can include in registration forms, where each field is uniquely identified by an *id*, with additional fields for the *field_name* and *field_type*, and optional *description* field provides further details about predefined fields.

Table 19

Event Registration Field

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each registration field
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
predefined_field_id	INT	NOT NULL, FK	Reference to the predefined field (predefined_registration_field.id)
required	BOOLEAN	NOT NULL	Indicator whether the field is mandatory

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Registration Field table links predefined fields to specific events. It contains two foreign keys*: event_id*, referencing the Event table, and *predefined_field_id*, referencing the Predefined Registration Field table. The required field indicates whether the field is mandatory.

Table 20

Registration Field Value

Column	Type	Constraints	Description
id	SERIAL	PK, NOT NULL	Unique identifier for each value
registration_id	UUID	NOT NULL, FK	Reference to the registration (registration.id)
event_registration_field_id	INT	NOT NULL, FK	Reference to the registration field (event_registration_field.id)
field_value	TEXT	NOT NULL	Value provided for the field

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Registration Field Value table stores user-provided data for custom registration linking a registration to a specific registration field via the *registration_id* and *event_registration_field_id* fields and *field_value* fields captures the user's response.

Table 21

Users Preference

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each preference
user_id	UUID	NOT NULL, FK	Reference to the user (users.id)
category_id	INT	NULL POSSIBLE, FK	Reference to the category (event_category.id)
tag_id	INT	NULL POSSIBLE, FK	Reference to the tag (event_tag.id)
keywords	TEXT	NULL POSSIBLE	Keywords related to preferences

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Users Preference table captures individual user preferences for personalization uniquely identifying each record by an *id* and links to Users, Category, or Tag tables via the *user_id*, *category_id*, and *tag_id* foreign keys. The keywords field allows for storing additional preferences not covered by predefined categories or tags.

Table 22

Users Interaction

Column	Type	Constraints	Description
id	INT	PK, NOT NULL	Unique identifier for each interaction
user_id	UUID	NULL POSSIBLE, FK	Reference to the user (users.id)
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
guest_id	UUID	NULL POSSIBLE	Reference to a guest user if applicable
interaction_type	VARCHAR (50)	NOT NUL	Type of interaction (e.g., like, share)
interaction_time	TIMESTAMP	NOT NUL	Timestamp of the interaction

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Users Interaction table logs user interactions with events where interaction itself is uniquely identified by an *id* and references a user or guest via the *user_id* or *guest_id* fields and an event via the *event_id* foreign key field. The *interaction_type* field captures the nature of the interaction (e.g., like, share), while the *interaction_time* field records when it occurred.

Table 23

Event Image

Column	Type	Constraints	Description
id	UUID	PK, NOT NULL	Unique identifier for each image
event_id	UUID	NOT NULL, FK	Reference to the event (events.id)
s3_key	UUID	NOT NULL, FK	Identifier for the image in S3 storage
is_private	UUID	NOT NULL, UNIQUE	Whether the image is private or public
uploaded_at	TIMESTAMP	NOT NULL	Timestamp of when the image was uploaded

Source: created by author by Dzmitry Papkou (2024) based on Relational Database Management System standards

The Event Image table stores metadata about images associated with events where each image is uniquely identified by an *id* and linked to an event via the *event_id* field. The *s3_key* specifies S3 cloud bucket storage location, while the *is_private* field determines whether the image is public or private and the *uploaded_at* field logs time when the image was uploaded.

4.8. Information processing, search, and retrieval

All information in “Event Goose” is processed according to privacy rules described in subchapter “1.2. Company's information management policy”, also before collecting and processing personal data that will be encrypted for security the system must obtain the consent of users. In addition, the user's system must provide the ability for the user to change their data such as password or email, username, etc., at any time.

For data processing will be used PostgreSQL, where SQL queries will allow sorting and filtering of data and search for the result required by the user by sending queries to the server, considering that the connections between the client and the server will be encrypted using HTTPS protocol, and sensitive personal data will be stored in mostly encrypted form for security. Then after processing, this data will be returned to the user in the form of a JSON response and displayed on the web page in a convenient form that will allow users to quickly find the necessary information to manage their events.

It is worth noting that lists of special offers for events like discounts will be available to all registered users based on their subscription plan, it should also be possible to search for services by

destination and filter results by various criteria, but it should be noted that only administrators will be able to modify or delete available by “Event Goose” offers. In case if offer was created by the external company with special request to the “Event Goose”, it also should be deleted or managed only by system administrators by special email request.

4.9. System testing and results evaluation

The testing environment was prepared as part of the predevelopment setup process, but actual test scripts were not implemented, as of the "Event Goose" application was conducted within the constraints of a tight development timeline, which significantly influenced the strategic decision to allocate the available time and resources to feature development and system design and to skip test automation. Moreover, since the platform being developed is not a commercial product, this removes the developer and owner's responsibility to the customer to ensure that the project is constantly tested under an automated environment.

This way unit tests and in similarly integration testing was only partially conducted, while no formal test cases were written, integration was estimation processed during the development process as a part of basic validation from development instruments and frameworks utilized. For instance, during the creation of features like user authentication and database operations, the interactions between the frontend, backend, cloud services and database were tested manually verifying that these components worked smoothly with each other and that no crashes and significant errors occurred.

Thus, as discussed earlier, manual testing was the primary method used to identify and correct potential problems. Interacted directly with the application by simulating real-world usage scenarios to identify flaws issues, which provided valuable insights of system work process that helped eliminate most of the obvious problems during the development phase.

In addition, database testing was partly solved by Liquibase, which was used to manage and test schema migration, ensuring versioning of changes to the database structure that prevented cause of conflicts and errors on project compilation and in continuous deployment. Although specific test cases for database queries and performance were not developed, informal checks confirmed that the database performed reliably under typical usage conditions. For example, adding, fetching, and modifying data through the application interface did not result in data failures or inconsistencies, indicating that the integration of the database and other components functioned as expected.

4.10. Conclusion on information system project

The system implementation of the "Event Goose" system following the framework defined in chapter 3, providing an overview on the functional system prototype. The backend, implemented using Java and the Spring Framework, and the frontend, developed with React and TypeScript, were integrated to what confirmed by integrated system operation between independent component, while the system's deployment on AWS, supported by containerization through Docker and a continuous integration pipeline provides production ready system.

Key components of the platform after development, including event management, secure data and payment processing, and database operations, were successfully implemented. The Liquibase-managed database structure is also done, allowing data handling and modifications integration of the features such as event tagging, location mapping, and notifications demonstrates the platform's ability to meet outlined requirements. That represents the practical realization of the project design, ensuring that the core functionalities of the system align with the initial objectives

CONCLUSIONS

1. The market analysis of event planning platforms such as “Eventbrite”, “Meetup”, and “Ticketmaster” revealed significant gaps in personalization, scalability, and feature integration. These findings were insightful and helpful enough in shaping the design of new information system by identifying the need for an all-in-one platform that integrates event creation, discovery, promotion, and registration. Later by addressing these gaps, "Event Goose" sets itself apart as a solution designed to meet evolving user expectations and industry standards, laying the groundwork for a competitive event management system in event planning market

2. The system integrates core functionalities, including event search, creation, promotion, and participant registration, providing a cohesive platform for managing events effectively. The backend, developed with Java and the Spring Framework, supported by PostgreSQL, ensures a strong foundation for any business logic operations, eliminating the need for users to rely on multiple tools, simplifying processes and enhancing efficiency in facilitates event planning and participation, and offering clear and organized pathways for users to manage events from start till end.

3. The implementation of the web interface delivered to users the gateway to see and managing various aspects of developed system through the browser, demonstrating core functionalities, such as event creation, registration, and discovery, with are accessible and efficiently integrated with backend systems to ensure smooth and consistent operations and data delivery. The implemented design for now prioritizes features integration to demonstrate abilities of the platform, making it more demonstrative than final implemented, what make it well-suited for future enhancements and expansion as user needs evolve.

4. The integration with AWS services for project deployment was established and deployed using Docker containers on AWS EC2 instances, where required static assets hosted on private and public S3 buckets and delivered globally via CloudFront for optimal performance scale. Amazon RDS provided a scalable database solution, while Cognito and SES were integrated as secure user authentication and efficient communication tools. As well, partially automated CI/CD pipeline was implemented, with a minimal manual intervention for image building align on security reasons of human based control over application deployment process.

5. The payment system model was introduced and conceptually implemented as a part of platform functionality, using a Stripe test environment. Although full implementation of payment integration remains a future task, the groundwork for a multi-currency and secure transaction framework has been established. This integration addresses requirements for user and business transactions, with plans for realization of proposed methods of platform monetization like freemium, subscription and referral model.

Recommendations

Future improvements for “Event Goose” should prioritize the integration of real-time data management by implementing a real-time server solution like Redis to handle dynamic updates for event registrations and participant interactions efficiently. Enhancing the frontend's optimization and adaptability will ensure consistent performance across all devices, including mobile platforms, broadening accessibility. Automating testing processes and writing a comprehensive suite of tests, including unit and integration tests, will significantly improve system reliability and maintainability. Additionally, completing the integration of the payment system and improve of the CI/CD pipeline specifically in the system versioning for production deployments will strengthen the platform's readiness for broader use.

REFERENCES

[1] Spring Framework Documentation. (n.d.). *Spring Framework Official Documentation*. Available at: https://spring.io/projects/spring-framework.

[2] React Documentation. (n.d.). *React Official Documentation*. Available at: https://reactjs.org/docs/getting-started.html.

[3] Java Documentation. (n.d.). *Java Official Documentation*. Available at: https://docs.oracle.com/javase/21/docs/.

[4] Material-UI Documentation. (n.d.). *MUI: React Component Library Documentation*. Available at: https://mui.com/material-ui/getting-started/overview/.

[5] Google API Documentation. (n.d.). *Google Cloud API Documentation*. Available at: https://cloud.google.com/apis/docs.

[6] AWS API Documentation. (n.d.). *AWS Official Documentation*. Available at: https://docs.aws.amazon.com/index.html.

[7] Allied Market Research. (n.d.). Events Industry Market. Available at: https://www.alliedmarketresearch.com/events-industry-market.

[8] Eventcube. (2024). Key Event Industry Statistics, Data, Trends, and Insights in 2024. Available at: https://www.eventcube.io/blog/key-event-industry-statistics-data-trends-and-insights-in-2024.

[9] Quadrant2Design. (n.d.). The Events Industry: Key Statistics. Available at: https://www.quadrant2design.com/the-events-industry-key-statistics/.

[10] Upmetrics. (n.d.). Event Planning Industry Statistics. Available at: https://upmetrics.co/blog/event-planning-industry-statistics.

[11] Eventbrite. (n.d.). *Event Management Platform*. Available at: https://www.eventbrite.com/blog/event-management-platform

[12] Eventbrite. (n.d.). *About Eventbrite*. Available at: https://www.eventbrite.com/.

[13] Meetup. (n.d.). *About Meetup: Connecting Communities*. Available at: https://www.meetup.com/.

[14] Ticketmaster. (n.d.). *Ticketmaster Official Website*. Available at: https://www.ticketmaster.com/.

[15] Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*.

[16] Cambridge University Press. Chapter 6: Scoring, Term Weighting, and the Vector Space Model, pp. 120–125.

[17] Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix Factorization Techniques for Recommender Systems*. IEEE Computer Society.

[18] Fowler, M., & Scott, K. (2000). *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley.

[19] Chen, P. P. (1976). *The Entity-Relationship Model: Toward a Unified View of Data*. ACM Transactions on Database Systems, 1(1), pp. 9–36. Yourdon, E. (1989). *Modern Structured Analysis*. Yourdon Press.

[20] Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *The Unified Modeling Language User Guide*. Addison-Wesley.

[21] Amazon Web Services. (2024). *AWS Architecture Icons and Guidelines*. Available at: https://aws.amazon.com/architecture/icons/.

[22] GitLab. (n.d.). *GitLab CI/CD Documentation*. Available at: https://docs.gitlab.com/ee/ci/.

[23] Codd, E. F. (1970). *A Relational Model of Data for Large Shared Data Banks*. Communications of the ACM, 13(6), pp. 377–387.

[24] Silberschatz, A., Korth, H. F., & Sudarshan, S. (2019). *Database System Concepts*. McGraw-Hill.

[25] European Parliament and Council of the European Union. (2016). Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data (General Data Protection Regulation). Official Journal of the European Union, L 119/1. Available at: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679

[26] Lopata and V. Moskaliova, (2017). *Methodological guidelines for professional practice and writing bachelor's thesis*.