Constructing A Software Requirements Specification Document and Prototype for Advanced Software Design Course

Unified Academic Event Platform for Berlin Universities

ABSTRACT

In this project, we present the design and development of a Unified Academic Event Platform tailored for Berlin universities.

Our approach follows **Feature-Driven Development (FDD), Behavior-Driven Development (BDD), Test-Driven Development (TDD), Risk-Driven Development (RDD), and Data-Driven Development (DDD)** to ensure robustness, scalability, and maintainability.

The system architecture is designed using **a layered model, finite state machines (FSM), UML diagrams, and entity-relationship models**, ensuring clear data flow and modular component interactions.

Security measures such as multi-factor authentication (MFA), API rate limiting, and AES-256 encryption **ensure compliance with GDPR standards**. A **risk-based feasibility study** guides system resilience, while **automated testing** (unit, integration, and end-to-end) guarantees reliability.

The document details a **KANBAN-driven product backlog, DevOps strategies, and a CI/CD pipeline for deployment**.

by

Aida Sharbatdar – Can (John) Özkan – Shayan Rhimi – Tarun Akkarakalam

Advanced Software Design Course – PW24 – Master of Software Engineering

Contents

TABLE OF TABLES

TABLE OF FIGURES

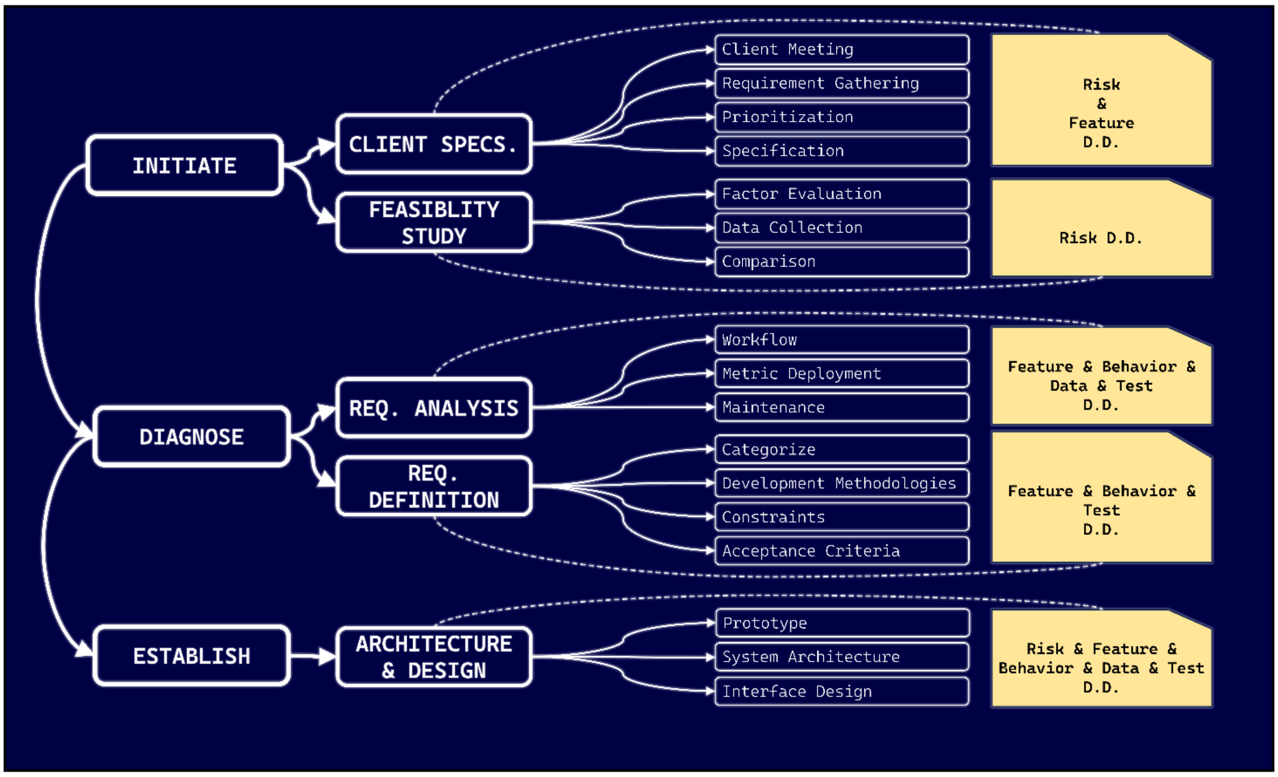

Figure 1 Project Framework

1. INITIATE

1.1. CLIENT SPECIFICATIONS

1.1.1. Client Meeting

1.1.1.1. The Main Goal:

To create a web-based educational events listing service that advertises academic talks and events across Berlin's universities to researchers, students, postgraduates, and other interested parties.

1.1.1.2. The End-Users:

The platform serves four distinct user categories with specific roles and permissions:

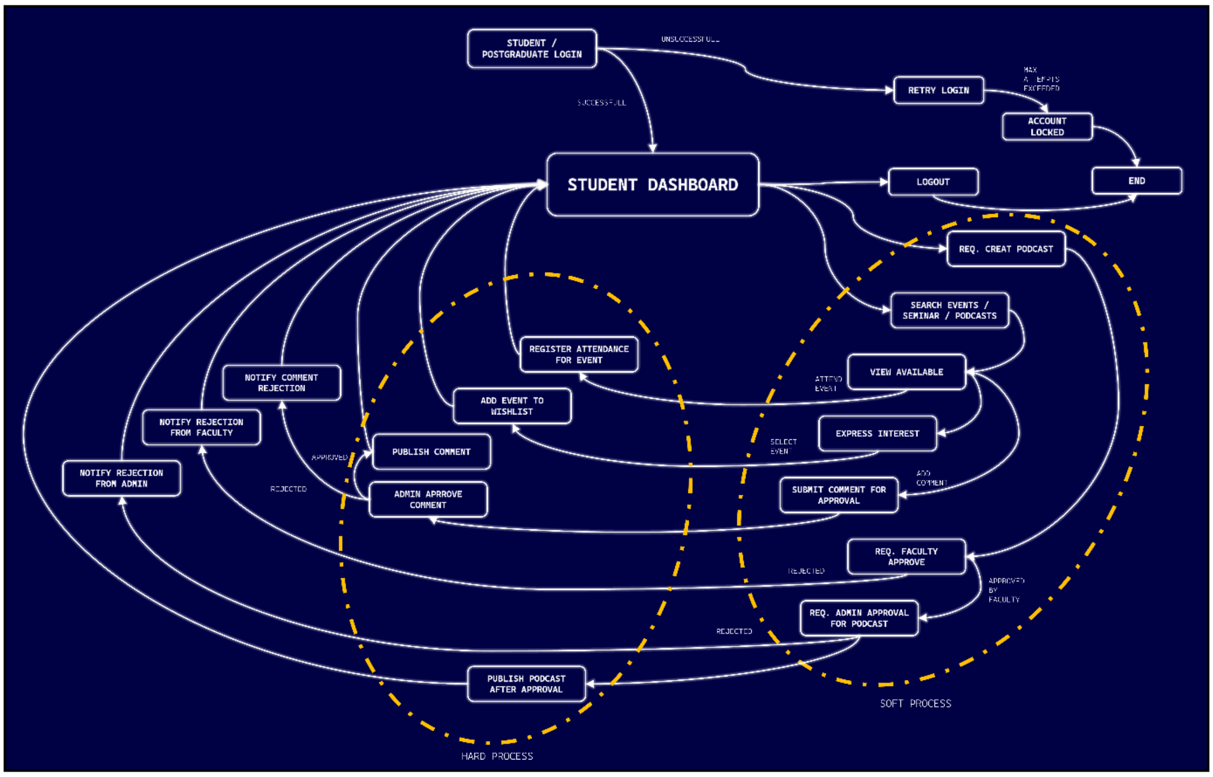

Figure 2 Student & Postgraduate Map

1. **Students and Postgraduates**
   - Access via university email credentials
   - Can attend events and create podcasts with faculty supervision
   - Full interaction privileges (comments, interest marking)

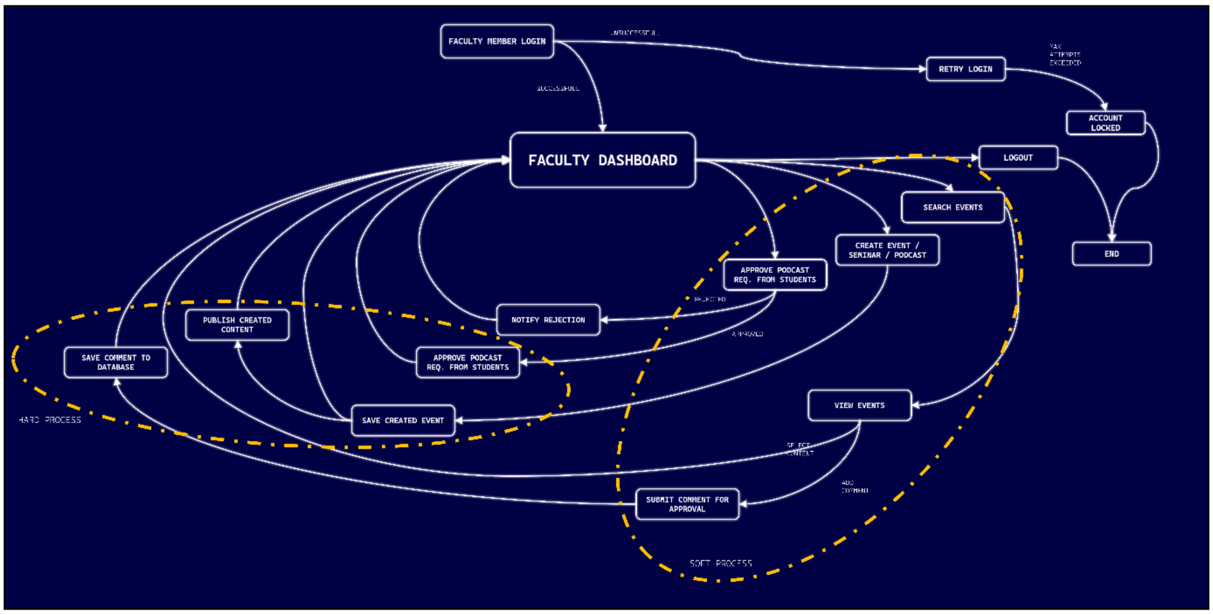

Figure 3 Faculty Member Map

2. **Faculty Members**
   - Full student privileges plus independent content creation

- Potential admin role if assigned by development team

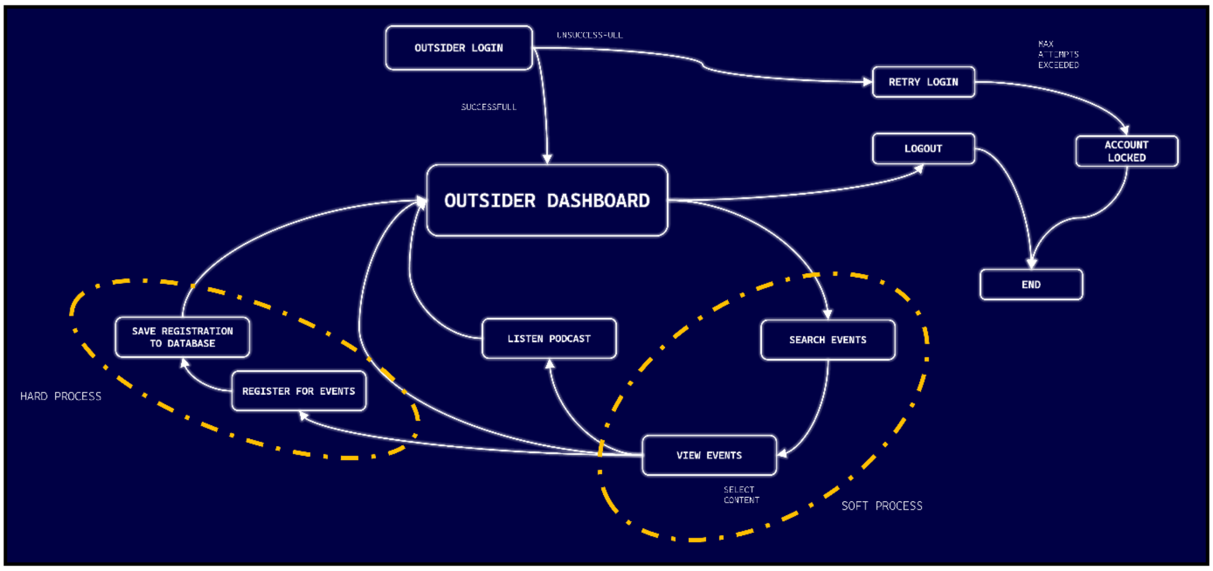

Figure 4 Outsider Map

3. **Outsiders (Non-University Users)**
   - Limited access without university email
   - Can attend events and access podcasts
   - Restricted from content creation and commenting

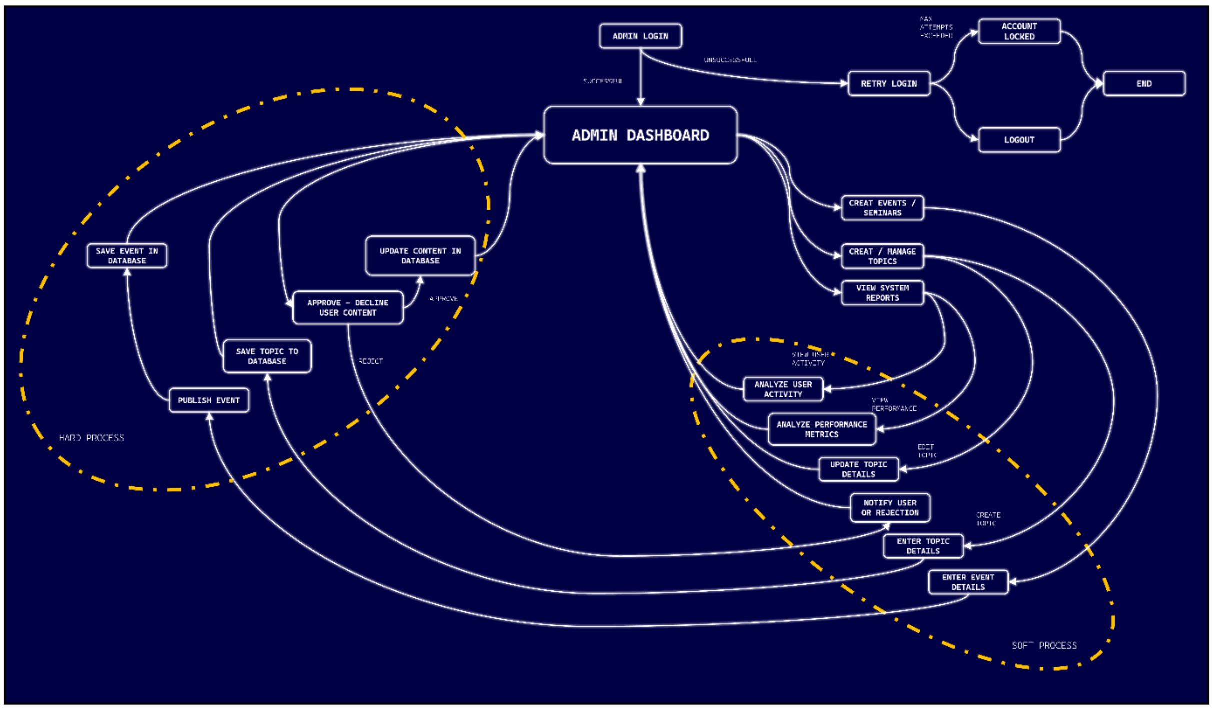

Figure 5 Admin Map

4. **Administrators**
   - One per faculty per university (maximum three across institutions)
   - Manage content approval and user permissions
   - Faculty-specific administration rights

1.1.1.3. The Problem to Be Solved:

It will address the challenge of discovering and accessing information about events and talks occurring across different university campuses in Berlin.

1.1.2. Requirement Gathering

1.1.2.1. Functional Requirements

1.1.2.1.1. Core Platform Functions - FDD

1. **Authentication and Access**
   - University email verification system
   - Role-based access control
   - Session management and security
2. **Content Management**
   - Event creation and publication
   - Podcast hosting and streaming
   - Seminar organization
   - Comment system with moderation
3. **User Interaction**
   - Content rating and feedback
   - Interest marking system
   - Attendance tracking
   - Navigation support

TABLE I. FEATURE ACCESS MATRIX

Feature	**Students & Postgraduates**	**Faculty Members**	**Outsiders**	**Admins**
Attend Seminars/Events	✓	✓	✓	✓
Listen to Podcasts	✓	✓	✓	✓
Comment on Features	✓	✓	✗	Approve/Decline
Create Podcasts	With Chaperone	✓ (Independent)	✗	Approve
Approve User Requests	✗	✗	✗	✓
Create Events/Seminars	✗	✓	✗	✓
Assign Topics to Events	✗	✗	✗	✓

Table 1 FEATURE ACCESS MATRIX

1.1.2.1.2. Specific Features or Functionalities:

The ability for users to create and subscribe to event lists, verify university affiliations, manage recurring and single events, and generate location and transportation details for events.

1.1.2.1.3. Particular Workflows to Automate:

Verifying university affiliations through email, creating and managing talk lists, and generating transport links for event locations.

1.1.2.1.4. Data Inputs and Outputs:

User data (e.g., emails for verification), event details (e.g., location, time, campus), and generated transportation links.

1.1.2.1.5. Specific Scenarios for Different Behavior:

Managing interest groups where only selected talks from a series are included in the advertised list.

1.1.2.2. Non-Functional Requirements

1.1.2.2.1. User Security and Data Protection:

Verification through university email and adherence to data protection standards.

1.1.2.3. Technical Requirements

1.1.2.3.1. Integration With Third-Party Applications:

Integration with TransLink and Google Maps for transport information.

1.1.2.4. Constraints

1.1.2.4.1. Specific Deadlines:

Feasibility study, requirement analysis and definition, architecture and design to be made till 12.01.2025

1.1.3. Prioritization

1.1.3.1. Must-Haves And Nice-To-Haves

1.1.3.1.1. The Absolute Essential Features:

Event creation, subscription to talk lists, university affiliation verification, and transport link generation.

1.1.3.2. Risk Assessment - RDD

TABLE II. INITIAL RISK ASSESSMENT

Risk ID	Risk Description	Potential Impact	Likelihood	Priority	Mitigation Strategy
R-001	Incorrect university affiliation verification	Unauthorized event creation	High	High	Implement automated email token-based verification.
R-002	System failure under high concurrent user load	Event platform becomes inaccessible	Medium	High	Optimize system architecture; perform load testing.
R-003	Privacy risks due to insecure data storage	Breach of sensitive user information	Medium	High	Encrypt sensitive data using HTTPS and AES-256.
R-004	Manual approval delays for event creation	Reduced user satisfaction	Low	Medium	Automate approval for non-critical events using metadata.
R-005	Duplicate events created by different users	Redundant data in the platform	Low	Low	Implement duplicate detection based on event metadata.

Table 2 INITIAL RISK ASSESSMENT

Risk-Based Prioritization

- High Priority:

  Secure user authentication and data protection.

  Scalability to handle high user loads.

- Medium Priority:

  Automated approval for non-critical events.

- Low Priority:

Duplicate event detection and prevention.

1.1.4. Specification & Document Creation

1.1.4.1. Scope:

None specified.

1.1.4.2. Detailed Requirements:

1. **Core Functionalities**
   a. **Seminars**
      - In-person and online attendance options
      - Real-time location directions
      - Virtual participation links
      - Attendance tracking
      - Recording capabilities

   b. **Events**
      - Location management
      - Attendance options
      - Schedule management
      - Interest tracking
      - Category assignment

   c. **Podcasts**
      - Hosting infrastructure
      - Streaming capabilities
      - Creation workflow management
      - Faculty supervision system
      - Quality control measures
2. **Sub-Features**
   a. **Content Discovery**
      - Subscribed topics
      - Personalized recommendations
      - Popular content tracking
      - Active events dashboard

   b. **User Management**
      - Profile management
      - Wishlist system
      - Interest tracking
      - Interaction history.

1.1.4.3. User Stories:

1.1.4.3.1. User Roles and Permissions:

Students and Postgraduates

- Access: Login via university email credentials to access seminars, events, and podcasts.

Participation:

- Attend in-person or online seminars and events.
- Listen to podcasts through the platform.

Content Creation:

- Create podcasts under faculty supervision. Requests require approval from a faculty member and subsequent admin validation.

Interactions:

- Comment, edit, or delete their comments across all core features.
- Express interest in specific seminars, events, or podcasts.

Navigation Support:

- View event and seminar directions; directions for podcasts are unnecessary due to in-app broadcasting.

Faculty Members

Permissions:

- Full student privileges, including attending, commenting, and marking interests.
- Create seminars, events, and podcasts independently. These can be tied to existing topics or categorized under new, independent themes ("floating features").
- Floating Topics: If a faculty member creates an event or seminar without an assigned topic, it becomes a floating event. Admins can view floating events and either assign them to an existing topic or create a new topic and assign the event to it.
- Administration: May act as admins if assigned by the development team.

Outsiders

Access:

- Register without a university email.
- Limited participation: attend and express interest in events, seminars, or podcasts.
- Restricted from commenting or creating content.

4. Admins

Roles:

- Appointed by the development team (one per faculty, up to three across institutions).
- Each faculty can have a minimum of one admin and a maximum of three admins. However, only one admin per faculty is permitted per university. For example:

  Faculty X can have one admin from University A, one from University B, and one from University C.

  If Faculty X has an admin from University A, it is not mandatory for Universities B or C to appoint admins for Faculty X.

Responsibilities:

- Approve or decline all user-generated content, including event requests and comments, for their assigned faculty.
- Create and manage topics and associated events.
- Assign faculty member status to eligible users.

Restrictions:

- Admins can only administrate the features or sub-features of their own faculty. For example:
- An admin of Faculty X at University A cannot control the features of Faculty Y or Z, regardless of the university.

1.1.4.3.2. Core Functionalities - FDD

Seminars

- Public lectures and academic talks that include:
- In-person and online attendance options.
- Real-time location directions and virtual participation links.

Events

- Themed social gatherings with features such as:
- In-person and online attendance.
- Accessible location and schedule details.

Podcasts

- Educational media available on-demand via the platform.

1.1.4.3.3. Sub-Features - FDD

Subscribed:

- Personalized list of topics and features chosen by the user.

For You:

- Recommendations based on the user's subscription and interaction history.

Other:

- Features not categorized under seminars, events, or podcasts.

Topic:

- Administrator-defined categories to organize events and features. Only admins can create topics.

Wishlist:

- A collection of features marked as "interested" by the user.

Active Events:

- A live dashboard of ongoing broadcasts.

Popular Events:

- Features with high attendance or interest metrics.

1.1.4.3.4. The Key Scenarios:

Student Content Creation

- Submit creation request
- Receive faculty approval
- Obtain admin validation
- Publish content

Faculty Event Management

- Create event independently
- Assign to topic or float
- Monitor attendance
- Gather feedback

Admin Topic Management

- Create new topics
- Assign floating content
- Moderate comments
- Manage permissions

1.1.4.3.5. Typical User's Goal:

To find and attend events and talks that match their academic or personal interests across Berlin's campuses.

1.1.4.3.6. Critical Pain Points of Users:

Difficulty in discovering and accessing information about events at other universities.

1.1.4.4. Acceptance Criteria:

Authentication Criteria

- Valid university email verification
- Role-appropriate access levels
- Secure session management

Content Management Criteria

- Successful content creation workflow
- Proper permission enforcement
- Accurate topic categorization

User Experience Criteria

- Intuitive navigation
- Responsive interface
- Clear error messaging

1.2. FEASIBILITY STUDY

1.2.1. Factor Evaluation

1.2.1.1. Technical

1.2.1.1.1. Technological Assessments:

- ***Do we have the right tools and technology to execute this project?***

  We have access to essential tools, including Figma for prototyping, Visual Studio for coding, and GitHub for version control and collaboration. These tools are industry standards and sufficient for the project's scope. However, we might consider integrating additional tools such as JIRA for task management and team coordination if needed.

- ***Are the required platforms or frameworks readily available or cost-effective?***

  Yes, all required platforms and frameworks, such as web development libraries (e.g., React, Node.js), are free or have affordable versions suitable for student projects.

- ***Can we leverage any existing systems or technologies?***

  While there's no immediate need to integrate existing systems, using public APIs (e.g., Google Maps API) enhances functionality without significant additional effort.

1.2.1.1.2. Team Skills:

- ***Does the team have the expertise needed for this project?***

  The team consists of three members experienced in web development (front-end and back-end) and one with strong business and team management skills. This diverse skill set ensures efficient progress at this level. However, periodic skill assessments may identify gaps for long-term scalability.

- ***Will additional training or hiring be required?***

  Not for the current scope. For scalability, the business expert may need technical training, and hiring additional members specializing in mobile development or cloud services could be considered.

- ***Are there any knowledge gaps that could slow down development?***

  Minor gaps exist, such as expertise in deploying large-scale systems or advanced cloud integrations. These gaps are manageable within the current timeline but may pose challenges if the project scales.

1.2.1.1.3. Integration:

- ***Can the new software integrate smoothly with existing systems?***

  The software is designed as a standalone system, eliminating integration challenges.

- ***Are there compatibility risks with third-party tools or APIs?***

APIs like Google Maps are publicly available. While budget constraints for API usage may arise during full implementation, this is irrelevant for a prototype.

- ***What are the potential challenges in data migration or synchronization?***

  As a standalone project, data migration and synchronization are not required, simplifying development.

1.2.1.1.4. *Scalability:*

- ***Can the system handle increased load or users in the future?***

  The architecture will be designed to support scalability by using load-balancing techniques and modular coding practices.

- ***Are there risks of performance degradation as the system grows?***

  Potential risks include slower load times or server overloads, mitigated by optimized database queries and caching strategies.

1.2.1.1.5. *Technical Risks:*

- **API Rate Limits**: Google Maps API may limit requests, impacting user experience.
  - **Mitigation**: Monitor usage and switch to premium API if necessary.
- **Compatibility Issues**: Issues with third-party integrations.
  - **Mitigation**: Perform thorough compatibility tests during development.

1.2.1.2. Operational

1.2.1.2.1. *User Acceptance:*

- ***How well does the proposed solution align with user needs?***

  The proposed solution addresses key user needs, including simplicity, speed, reliability, and privacy.

- ***Are there foreseeable resistance points from end-users?***

  Resistance points might include:

  - Long login processes.
  - Complex interfaces.
  - Perceived tracking or data collection.

  These can be mitigated through clear communication of data privacy policies and intuitive design.

1.2.1.2.2. *Training Requirements:*

- ***Will the users require significant training to adopt the new system?***

  The user interface (UI) will be designed to be self-explanatory, ensuring minimal need for training.

- ***Do we have the resources to provide adequate training and support?***

Since training is unnecessary, this resource allocation is not required.

1.2.1.2.3. Cultural Fitness:

- ***Does the solution align with the client's organizational culture?***

  Yes. The app includes admin controls to ensure alignment with each university's policies. Additionally, its accessibility ensures usability across diverse age groups.

  Since the clients are 3 separate universities, their brand identities differ but the app will primarily for the people aging between 18-35 as they are students and post-graduates.

  The app will be easy to navigate so the faculty members who are in the age group of 40-80 can also be able to use.

1.2.1.2.4. Operational Risks:

- **User Resistance**: Complexity in user interfaces may deter engagement.
  - **Mitigation**: Conduct usability testing and simplify the design.
- **Onboarding Delays**: Manual verification slows down the process.
  - **Mitigation**: Introduce automated, token-based verification processes.

1.2.1.3. Economical

1.2.1.3.1. Cost Analysis:

- ***What are the estimated costs for development, deployment, and maintenance?***

  As a school project, costs are negligible. If fully implemented, costs include API fees, cloud storage, and app store publishing fees.

- ***Are there hidden costs we might overlook, such as licensing or hardware upgrades?***

  Potential hidden costs include scaling cloud services, API rate limits, and additional developer tools for team expansion.

1.2.1.4. Schedule

1.2.1.4.1. Timeline Assessment:

- ***Can the project be completed within the proposed timeframe?***

  The proposed timeline is achievable:

  Client specifications: 12.12.24

  Feasibility study: 22.12.24

  Requirements analysis: 29.12.24

  Architecture design: 07.01.25

  Final presentation: 11.01.25

- ***What are the risks if delays occur?***

  Missing deadlines jeopardizes project completion and could lead to academic penalties.

1.2.1.5. Legal

1.2.1.5.1. Regulations:

- ***Are there specific regulations or standards we must comply with?***

  The system will adhere to GDPR, ensuring user data protection and privacy.

1.2.1.5.2. Ethical Concerns:

- ***Does the solution raise any ethical questions or concerns?***

  The system avoids ethical concerns by not storing sensitive user data and adhering to regulatory standards.

1.2.1.6. Risk Strategies & Analysis - RDD

TABLE III. RISK MITIGATION STRATEGIES

Risk Type	**Identified Risks**	**Mitigation Strategy**
Technical	API rate limits	Monitor usage; evaluate and switch to premium plans.
Technical	Compatibility issues	Conduct integration and compatibility testing.
Operational	User resistance to complex UI	Simplify interfaces; perform usability testing.
Operational	Onboarding delays	Automate verification; provide faster token validation.

Table 3 RISK MITIGATION STRATEGIES

TABLE IV. RISK-BASED COST-BENEFIT ANALYSIS

Risk Type	**Risk Description**	**Associated Cost**	**Mitigation Cost**	**Net Benefit**
Technical	API rate limits impacting user experience	Increased API fees	Premium API plan fees	Improved user satisfaction.
Operational	Delays in user onboarding	Lower user retention	Automated verification	Faster onboarding; happier users.
Business	Budget overruns due to scaling	High infrastructure	Cloud optimization costs	Scalable, reliable platform.
Risk Type	Risk Description	Associated Cost	Mitigation Cost	Net Benefit

Table 4 RISK-BASED COST-BENEFIT ANALYSIS

1.2.2. Data Collection

1.2.2.1. Interviews:

- ***What insights can we gain from stakeholders or end-users?***

  Stakeholders prioritize simplicity, speed, and data security. Addressing these points ensures project success.

1.2.2.2. Market Research:

- ***How do similar projects or solutions perform in the market?***

  Similar platforms are popular in Germany for event promotion, highlighting a demand for such solutions.

1.2.3. Comparison

1.2.3.1. Proposed Solutions:

- ***What are the possible approaches to achieve the project's goals?***

  Following the professor's instructions:

  - Develop a Software Requirements Specification (SRS) document.
  - Use the IDEAL model for risk and data-driven development.
  - Include domain mapping, user-function relations, data flow diagrams, and entity-relationship models.
  - Implement a test plan for verification and validation.
  - Standardize QA with LoQ benchmarking, CICD pipelines, FSM, and UML diagrams.
  - Design UI/UX prototypes and implement the API layer.
  - Create a functional checklist.

1.2.3.2. Pros & Cons:

- ***What are the strengths and weaknesses of each proposed solution?***
  - **Strengths**:
    - Flexibility: The system is designed independently, requiring no integration with external systems, which simplifies development and minimizes risks.
    - Cost-Effectiveness: Freely available tools (Figma, Visual Basic, GitHub) and no additional hardware or licensing costs make the project financially feasible.
    - Team Expertise: A diverse skill set within the team ensures effective collaboration and division of tasks.
    - Focus on User Needs: The system's simplicity, speed, and reliability directly align with user expectations.
  - **Weaknesses**:
    - Limited Budget: The lack of funding might restrict access to advanced tools, APIs, or services that could improve quality.
    - Knowledge Gaps: While manageable within the project's scope, potential scaling challenges may arise if the project expands.
    - Limited Market Research: A deeper understanding of similar products and user preferences is constrained by resource limitations.
    - Communication Barriers: Team members from different linguistic backgrounds might face occasional misunderstandings.
- ***Are there any trade-offs between cost, time, and quality?***

  Cost vs. Quality: The reliance on free tools and limited resources may result in certain compromises, such as basic features or reduced scalability. However, the team's expertise mitigates this risk by optimizing quality within constraints.

  Time vs. Quality: The tight timeline ensures the project remains on track but may limit opportunities for thorough testing or refinement.

  Cost vs. Time: The absence of a budget prevents outsourcing or additional hiring, which could have accelerated development.

1.2.3.3. Best Option:

- ***Which solution is the most viable considering all the feasibility factors?***

  The proposed solution—developing a standalone system using free tools and leveraging the team's existing expertise—is the most viable. It aligns with resource availability, user needs, and project constraints. The step-by-step adherence to the professor's outlined requirements ensures the project remains feasible within the given timeframe.

- ***What risks remain even with the best option, and how can they be managed?***

  Performance Risks: Scalability and load handling might become issues in a real-world scenario. Mitigation: Implement robust testing for performance under simulated high-usage conditions.

  Knowledge Gaps: The business-skilled team member may face challenges in technical tasks if required. Mitigation: Allocate tasks based on strengths and provide focused training if necessary.

API and Licensing Risks: If the project scales, API costs and licensing fees (e.g., Google Maps, AWS) could arise. Mitigation: Conduct a cost analysis and explore cost-efficient alternatives if scaling becomes necessary.

Communication Challenges: Potential misunderstandings due to linguistic diversity. Mitigation: Schedule regular team check-ins and ensure clarity in documentation and task delegation.

Stakeholder Misalignment: Limited feedback from client stakeholders could lead to unmet expectations. Mitigation: Regularly review client specifications and proactively address gaps or ambiguities in requirements.

2. DIAGNOSE

2.1. Requirement Analysis

2.1.1. Workflow

2.1.1.1. Dataflow

2.1.1.1.1. Information Processed from Start to Finish:

- The platform will serve as a web-based service for listing educational and industrial events across three or more major university campuses in Berlin.
- Users can search for, view, and register for events.
- Search and register system to process **1.000+** con-current user requests.
- Verified university-affiliated users can create, manage, and advertise events, talks, or talk series.
- Administrators manage event lists for specific interest groups.
- The system generates location-based transport links to aid event attendees.

2.1.1.1.2. Data Inputs Required - DDD:

User Data

- University email credentials
- Role-specific information
- Interaction history
- Preferences

Content Data

- Event details
- Location information
- Topic categorization
- Approval status

System Data

- Access logs
- Performance metrics
- Error reports
- Usage statistics

TABLE V. DATA REQUIREMENT MATRIX

Data Type	**Required Fields**	**Validation**	**Storage Duration**
User Profile	Email, Role, Faculty	Email Verification	Account Lifetime
Content	Title, Description, Type	Admin Approval	Content Lifetime
Interaction	User ID, Content ID, Type	Permission Check	1 Year
System Logs	Timestamp, Action, Status	Automated	90 Days

Table 5 DATA REQUIREMENT MATRIX

2.1.1.1.3. Processes or Actions on Inputs:

- Validation of user credentials
  - University email verification.
- CRUD operations for events (Create, Read, Update, Delete).
- Filtering and searching for events
  - by category, time, or location.
- Linking location data with external transport services.
- Administrator approval for certain events in interest groups.

2.1.1.1.4. Data Flow Between Components:

- User inputs are processed and stored in the database.
- Search queries fetch data from the database to display results.
  - **95% under 100ms**
- Transport APIs interact with user input to generate travel information.
- Administrative actions trigger updates to the event database.

2.1.1.1.5. Triggers for Each Process:

- User sign-up triggers verification.
- Event creation or update triggers database updates.
- Search requests trigger retrieval of filtered data.
- Admin approvals trigger updates to advertised events.

2.1.1.1.6. Workflow Bottlenecks and Redundancies:

- Bottleneck:
  - Verification of university affiliation (manual process may slow onboarding) with a maximum delay of 4 hours for 95% of cases.
- Redundancy:
  - Event duplication if admins create similar events, less than 2% overlap tolerance.

2.1.1.1.7. Behavior Specifications And Feature Breakdown – BDD & FDD

TABLE VI. BEHAVIOR SPECIFICATIONS

Behavior ID	**Given** **[Initial state or precondition]**	**When** **[Action performed by the user]**	**Then** **[Expected outcome or result]**
B-001	A user with a valid university email	They attempt to register	They receive a verification email.
B-002	A verified user	They create an event	The event is listed and visible.
B-003	A non-verified user	They attempt to create an event	They are prompted to verify their email.

Table 6 BEHAVIOR SPECIFICATIONS

TABLE VII. FEATURE BREAKDOWN

Domain	**Feature Name**	**Dependencies**
User	User Registration	None
User	Email Verification	User Registration
Event	Event Creation	Email Verification
Event	Event Management	Event Creation

Table 7 FEATURE BREAKDOWN

Feature Hierarchies:

- User Features:
  - Registration
  - Verification
- Event Features:
  - Creation
  - Management
  - Categorization
- System Features:
  - API Integration
  - Transport Link Generation
- Feature Dependencies
  - User Registration is a prerequisite for Email Verification.
  - Email Verification must be completed before Event Creation.
  - Event Creation enables Event Management and Transport Link Generation.

2.1.1.2. Error Handling

- Display clear error messages for invalid inputs (e.g., invalid email or missing fields).
- Log errors for backend processes (e.g., API failures).

2.1.1.3. Roles and Permissions - BDD

Permission Hierarchy

Admin Level

- Full content management
- User role assignment
- Topic creation/management
- Faculty-specific control

Faculty Level

- Independent content creation
- Supervision capabilities
- Comment management
- Event organization

Student Level

- Supervised content creation
- Full participation rights
- Comment capabilities
- Interest marking

Outsider Level

- Content viewing
- Event attendance
- Limited interaction

2.1.1.4. UX

- Intuitive navigation with search and categories (interests).
- Quick access to event details and registration options.
- Show directions to the event using google maps API.
- Showcasing recommended events (For you)

2.1.2. Metrics Deployment

2.1.2.1. Track

2.1.2.1.1. Key Metrics Definition

System Performance:

- Speed:

  Track the time taken for search queries, event registration, and API responses, **95% of queries to be estimated under 200ms.**

- Uptime:

  Monitor server availability to ensure minimal downtime.

- Response Time:

  Measure the latency of user requests, plan **for 90% of requests under 500ms.**

User Activity:

- Clicks:

  Count the number of clicks on events, filters, and navigation links which can be used to recommend user content, **track 100% of user interactions.**

- Task Completion Rates:

  Track how many users successfully register for events or complete key actions.

- Engagement:

  Measure the time spent by users on the platform.

  Expected activity per session per user targeted to be **8-10 minutes**.

- Retention:

  80% monthly active user retention.

  - *Phase 1 (Month 1-3):*

    Achieve **50%** retention through initial onboarding campaigns.

  - Phase 2 (Month 4-6):

    Reach **70%** by analyzing and improving user engagement.

  - Phase 3 (Post-Month 6):

    Sustain and **exceed 80%** retention with personalized recommendations and continuous feature updates.

System Resource Usage:

- Memory Usage:

  Calculate on and off-peak usage.

  Peak usage under 85% of allocated resources

- CPU Utilization:

  Target < 75%.

- Benchmarking:

  Conduct stress tests to establish baseline utilization.

- Optimization:

  Introduce caching strategies and optimize database queries to reduce processing load.

- Database Performance:

  Measure query performance, read/write speeds, and data retrieval times, utilize a fast and reliable database source (AWS, MongoDB, Firebase) to achieve query **response times of 95% under 100ms, write operations of 99% under 200ms and read operations of 99% under 50ms.**

Security Measures:

- Enhanced User Authentication:

  Implement multi-factor authentication (MFA) using email and mobile verification.

  Use hashed passwords stored securely with modern encryption algorithms like bcrypt.

- Data Encryption:

  Encrypt all sensitive user data (e.g., email, event details) in transit (using HTTPS) and at rest (using AES-256).

- API Security:

  Implement API rate limiting to prevent abuse.

  Log and monitor all API activity for suspicious patterns.

UX:

- Accessibility and Responsiveness:

  Ensure the platform is WCAG 2.1 compliant for accessibility.

  Test the interface on various devices (e.g., desktops, tablets, mobile phones) and browsers to ensure consistent performance.

- Usability Testing:

  Conduct A/B testing for layout and navigation flow.

  Include diverse user groups in usability tests to ensure inclusivity.

TABLE VIII. TRACING MATRIX FOR TRACKING

Requirement ID	Requirement Description	Metrics/Target	Validation/Monitoring
SP-01	Measure query speed	95% of queries under 200ms	Analyze API and database query logs
SP-02	Ensure server uptime	Minimal downtime	Uptime monitoring tools (e.g., CloudWatch, Pingdom)
SP-03	Measure request response time	90% of user requests under 500ms	Latency monitoring tools
UA-01	Track user clicks	100% of interactions captured	Log analysis for event tracking
UA-02	Monitor task completion rates	Successful key action completion	Analytics dashboards
UA-03	Track user engagement	Session time target: 8-10 minutes	Session activity tracking
UA-04	Improve user retention	80% monthly retention, phased goals: 50%, 70%, and >80%	Retention tracking reports
SR-01	Optimize memory usage	Peak usage under 85% of allocated resources	Resource monitoring tools
SR-02	Optimize CPU usage	Target <75%	CPU monitoring dashboards
SR-03	Enhance database performance	Query: 95% <100ms, Write: 99% <200ms, Read: 99% <50ms	Database performance monitoring
SEC-01	Implement user authentication	MFA with hashed passwords	Authentication audit logs
SEC-02	Encrypt sensitive user data	Data encrypted in transit (HTTPS) and at rest (AES-256)	Security compliance tools
SEC-03	Ensure API security	API rate limiting, logging, and anomaly detection	API activity monitoring
UX-01	Ensure accessibility and responsiveness	WCAG 2.1 compliance, consistent performance across devices and browsers	Accessibility and responsiveness testing
UX-02	Test usability	Conduct A/B testing, include diverse user groups	Usability testing reports

Table 8 TRACING MATRIX FOR TRACKING

SP: System Performance

UA: User Activity

SR: System Resource Usage

SEC: Security Measures

UX: User Experience

2.1.2.2. Detect

2.1.2.2.1. Issues or Pattern Identification

Bottleneck detection:

- *Verification Delays*:

  University affiliation verification delays are identified as a potential bottleneck.

- *Proposed Solution:*

  Implement an automated email verification system using a secure token mechanism. Users receive a token via their university email, and upon submission, the system validates it in real time, reducing manual delays.

- *Performance Target*:

  95% of verification requests processed within 5 minutes.

Redundancies:

- *Duplicate Event Creation:*

  Event duplication by administrators or users is identified as a redundancy.

- *Proposed Solution:*

  Introduce automated duplicate detection using event metadata comparison (e.g., title, time, location). The system will alert the creator of a potential duplicate and provide options to merge or override the event.

- Overlap Tolerance:

  Set a threshold for acceptable similarity **(90% match)** to trigger duplicate warnings.

Error Handling

- *Categorizing and Prioritizing Errors:*

  Implement a tiered error classification system:

  - **Critical**: Errors causing system outages (e.g., API failures).
  - **High**: Errors impacting key functionality (e.g., login failures).
  - **Medium**: Minor functionality issues (e.g., UI glitches).
  - **Low**: Cosmetic issues (e.g., typographical errors).

Recovery Actions:

For common user errors:

Display contextual hints and suggested fixes (e.g., "Invalid email format. Please ensure you use your university email address.").

Allow users to retry actions without losing progress.

For backend errors:

Retry failed operations (e.g., API calls) up to **3 times** before alerting the admin.

Maintain a fallback mode with minimal features for core functionalities.

- *Risk Mitigation:*

  Detect and block brute force login attempts with IP blacklisting and account lockout mechanisms.

  Monitor API usage for suspicious patterns and trigger alerts on anomalies.

- *Issue Detection:*

  Continuously monitor for unusual login attempts or unauthorized data access.

- *Enhancing Inclusivity:*

  Provide language options for non-German-speaking users.

  Offer adjustable font sizes and contrast settings for users with visual impairments.

2.1.2.2.2. Data Retention Policies

Detailed Logs:

90 days.

Protocols for archiving or deleting old data to comply with GDPR regulations.

Summary data:

365 days retention with 100% compliance.

2.1.2.2.3. Real time vs Batch Processing

Real-Time:

Performance metrics:

Updated every 30 seconds with 99% accuracy

User activity:

Processed with latency under 1 second for 95% of cases.

Batch Processing:

Weekly trend analysis with **98% accuracy.**

98% accuracy in trend identification

Error log summarization with **99% consistency**.

2.1.2.2.4. *Alert System*

Set thresholds for generating alerts:

System Downtime:

Trigger alerts if uptime falls below 99.9%.

Performance Degradation:

Notify admins if response time exceeds predefined limits.

Resource Usage:

Send alerts when CPU or memory usage crosses safe thresholds.

Monitor MTTR (Mean Time to Repair) and MTTF (Mean Time to Failure) to track maintenance efficiency.

MTTR target:

< 30 minutes for 90% of issues

MTTF target:

> 720 hours for 95% of components

TABLE IX. TRACING MATRIX FOR DETECTING

Requirement ID	Requirement Description	Metrics/Target	Validation/Monitoring
BOT-01	Reduce verification delays	95% processed within 5 minutes	Token verification system logs
BOT-02	Detect duplicate events	Metadata similarity threshold: 90%	Event metadata comparison logs
ERR-01	Classify and prioritize errors	Tiered system (Critical, High, Medium, Low)	Error logs and incident reports
ERR-02	Improve recovery actions for user/backend errors	Retry up to 3 times; fallback mode for core functionalities	User feedback and backend logs
ERR-03	Mitigate risks	Block brute force attempts, monitor API usage for anomalies	Anomaly detection and IP blocking logs
INC-01	Enhance inclusivity	Provide multilingual support and adjustable settings	User feedback and accessibility tests
DRP-01	Retain detailed logs	90 days	Periodic log audits
DRP-02	Retain summary data	365 days, GDPR compliance	Data retention compliance audits
PRC-01	Real-time performance metric updates	Update every 30 seconds with 99% accuracy	Real-time monitoring dashboards
PRC-02	Real-time user activity processing	Latency under 1 second for 95% of cases	User activity logs
PRC-03	Weekly batch trend analysis	98% accuracy	Trend analysis reports
ALRT-01	System downtime alerts	Trigger below 99.9% uptime	Downtime monitoring tools
ALRT-02	Performance degradation alerts	Notify if response time exceeds predefined limits	Performance monitoring dashboards
ALRT-03	Resource usage alerts	Alert if CPU/memory usage exceeds thresholds	Resource monitoring tools
ALRT-04	Track MTTR and MTTF	MTTR <30 mins (90%), MTTF >720 hours (95%)	Incident response reports

Table 9 TRACING MATRIX FOR DETECTING

BOT: Bottleneck Detection

ERR: Error Handling

INC: Inclusivity Enhancements

DRP: Data Retention Policies

PRC: Processing Modes

ALRT: Alert System

2.1.2.3. Report / Debugging

2.1.2.3.1. Timeframe:

- Real-time alerts:

  99.9% delivered within 1 minute

- Daily reports:

  Generated within 1 hour of day end (01:00 CET next day) with 100% accuracy

- Weekly reports:

  Completed by the start of the week.**(Monday – 6:00 am CET)**

2.1.2.3.2. Contact:

- First response time:

  < 5 minutes for 95% of critical issues

- Resolution time:

  < 2 hours for 90% of critical issues

2.1.2.3.3. Type of Reports:

- Dashboards:

  Provide a real-time overview of system metrics.

2.1.2.3.4. Alerts:

- Instant notifications for critical failures or breaches.
- Periodic Reports:

  Summarize system performance, user engagement, and resource usage trends.

2.1.3. Maintenance

2.1.3.1. Re-engineering

Code Quality Metrics:

- Code review coverage: **95% of new code**
- Technical debt reduction: **15% quarterly improvement**
- System efficiency improvement: **10% quarterly target**

Continuous Improvement Practices

Regular Architecture Reviews:

- Conduct monthly architecture assessments
- Identify potential bottlenecks and scalability issues
- **Target: Review 100% of critical system components quarterly**

Performance Optimization

Database Query Optimization:

- Implement query caching mechanisms
- Optimize database indexes
- **Target: Reduce query response time by 20% each quarter**

Code Refactoring Priorities

Legacy Code Modernization:

- Update deprecated dependencies
- Convert synchronous operations to asynchronous where beneficial
- **Target: Modernize 25% of legacy code each quarter**

Scalability Improvements

Infrastructure Scaling:

- Implement auto-scaling mechanisms
- Optimize resource utilization
- **Target: Support 30% increase in concurrent users per quarter**

Security Enhancements

Regular Security Audits:

- Conduct vulnerability assessments
- Update security protocols
- **Target: Monthly security audits with 100% critical vulnerability resolution**

2.1.3.2. Coding Standards

Naming Conventions

- Use "_" (underscore) for separation.
- Start every naming word with capital letter.
- Avoid using numbers if possible.
- Variables: Use descriptive names that convey the purpose.

  Example: `user_Age, total_Amount`.

- Functions: Use verbs or verb phrases.

  Example: `calculate_Total(), fetch_User_Data().`

- Classes: Use nouns or noun phrases.

  Example: `Invoice_Manager, User_Controller`.

- Constants: Use all uppercase letters with underscores.

  Example: `MAX_USERS, DEFAULT_TIMEOUT`.

- Files: Use lowercase letters with underscores.

  Example: `user_profile.js, invoiceManager.py`.

Formatting

- Indentation: Use 4 spaces per level.
- Line Length: Limit lines to 100 characters.
- Spacing:
  - Use spaces around operators and after commas.
  - Do not leave trailing whitespaces at the end of lines.
  - Add a newline at the end of the file.
- Braces: Place opening braces on the same line as the statement.

```
Example:
if (condition) {
// code
} else {
// code
}
```

Comments

- Block Comments: Use for complex logic and function descriptions.

```
/*
 * This function calculates the total amount
 * by adding tax and discount to the base
amount.
 */
```

- Inline Comments: Use sparingly for non-obvious code.

```
total += tax;  // Add tax to the total
```

- Documentation Comments: Use for public APIs and functions with detailed descriptions of parameters and return values.

```
/**
 * Calculates the total amount.
 *
 * @param base_Amount The base amount
 * @param tax The tax amount
 * @param discount The discount amount
 * @return The total amount after adding tax
and subtracting discount
 */
```

```
function calculate_Total(base_Amount, tax,
discount) {
    // code
}
```

Functions

- Single Responsibility: Each function should perform a single task.
- Function Length: Keep functions short and focused, ideally under 20 lines.
- Parameter Count: Limit the number of parameters to 3 or 4. Use objects to group related parameters if needed.

Error Handling

- Exceptions: Use exceptions for error handling, not for regular control flow.
- Validation: Validate inputs early and fail fast if something is wrong.
- Logging: Log errors with meaningful messages to help with debugging.

Code Structure

- Modularity: Break down large codebases into smaller, manageable modules.
- DRY Principle: Avoid code duplication by abstracting common logic into functions or classes.
- Separation of Concerns: Keep different concerns (e.g., data access, business logic, presentation) separated into distinct layers or modules.

Testing

- Unit Tests: Write unit tests for critical functions and components.
- Test Coverage: Aim for high test coverage without compromising the quality of tests.
- Naming Tests: Use descriptive names for test functions to indicate their purpose.

Version Control

- Commit Messages: Write clear and concise commit messages that explain the changes.
- Branching: Follow a branching strategy like GitFlow to manage feature development and releases.

Code Reviews

- Peer Reviews: Conduct regular code reviews to ensure adherence to coding standards.
- Automated Tools: Use static analysis tools to catch common issues and enforce coding standards.

Documentation

- Inline Documentation: Document key functions, classes, and modules within the code.
- API Documentation: Generate and maintain API documentation for public interfaces.

- Project Readme: Provide a readme file with setup instructions, usage examples, and contribution guidelines.

2.1.3.3. Documentation Structure

Project Documentation

- Title Page: Project name, version, date, and author(s).
- Table of Contents: An automatically generated table of contents for easy navigation.
- Introduction: A brief overview of the project, its objectives, and its key features.
- Setup Instructions: Detailed steps for setting up the development environment and installing dependencies.
- Usage Guide: Instructions on how to use the software, including examples and screenshots if applicable.
- API Documentation: Comprehensive details on the available APIs, endpoints, parameters, and responses.
- Architecture Overview: A high-level overview of the system architecture, including diagrams if necessary.
- Development Guidelines: Standards for coding, version control, and other development practices.
- Testing Procedures: Guidelines for writing and running tests, including test coverage expectations.
- Deployment Guide: Steps for deploying the software, including any environment-specific configurations.
- Troubleshooting: Common issues and their solutions.
- Contribution Guide: Instructions for contributing to the project, including code of conduct and pull request guidelines.

Writing Style

- Clarity: Write in a clear and concise manner. Avoid jargon and explain technical terms when necessary.
- Consistency: Use consistent terminology, formatting, and structure throughout the documentation.
- Tone: Use a professional yet approachable tone. Be respectful and inclusive in your language.

Formatting

- Headings: Use headings to organize content into sections and subsections.
- Lists: Use bullet points or numbered lists to present information in a readable format.
- Code Blocks: Use code blocks to display code snippets, ensuring they are properly formatted and syntax-highlighted.
- Tables: Use tables to organize data and present comparisons clearly.
- Images and Diagrams: Include relevant images, diagrams, and screenshots to aid understanding. Ensure they are labeled and referenced appropriately.

Version Control

- Commit Messages: Write descriptive and concise commit messages that explain the purpose of the changes.

- Change Log: Maintain a change log that records all significant updates to the documentation.

Review and Maintenance

- Peer Reviews: Conduct regular peer reviews of documentation to ensure accuracy and quality.
- Updates: Keep documentation up to date with the latest changes in the project.
- Feedback: Encourage feedback from users and contributors to improve the documentation.

Appendix

- Glossary: Include a glossary of terms used in the documentation.
- References: Provide links to additional resources and external documentation.
- Contact Information: Offer contact details for further assistance or inquiries.

2.1.3.4. Automated Testing - TDD

Test Coverage Requirements

- Unit Tests: **90% code coverage**
- Integration Tests: **85% coverage of critical paths**
- End-to-End Tests: **75% coverage of user workflows**

Testing Frameworks

Frontend Testing:

- Jest for unit testing
- Cypress for E2E testing
- **Target: 100% test automation for critical user paths**

Performance Testing

Load Testing:

- Apache JMeter for concurrent user simulation
- **Target: Successfully handle 1,000+ concurrent users**

Response Time Testing:

- **95% of requests under 200ms**
- **99.9% of requests under 500ms**

Continuous Integration Testing

Pre-commit Hooks:

- Lint checking
- Unit test execution
- **Target: 100% passing tests before merge**

Automated Build Pipeline:

- Build verification
- Integration test execution

- **Target: < 15 minutes total pipeline execution**

2.1.3.5. Version Control

Branch Strategy

Main Branches:

- main: Production code
- develop: Development integration
- **Target: 100% reviewed code in main branch**

Feature Development

Branch Naming:

- Format: feature/[ticket-number]-brief-description
- **Target: 100% compliance with naming convention**

Code Review Process

Pull Request Requirements:

- Minimum 1 reviewer approval
- All automated tests passing
- No critical security issues
- **Target: 95% of PRs reviewed within 24 hours**

Merge Criteria

- Clean build status
- All tests passing
- Code review approval
- No merge conflicts
- **Target: 0 failed deployments due to merge issues**

Version Tagging

Semantic Versioning:

- Major.Minor.Patch format
- Release tags for production deployments
- **Target: 100% tagged releases**

Backup and Recovery

Repository Backups:

- Daily automated backups
- **Target: 99.99% backup success rate**

Recovery Testing:

- Monthly recovery drills
- **Target: < 1 hour recovery time**

2.2. Requirement Definition

2.2.1. Categorize

2.2.1.1. Functional

TABEL X. FUNCTIONAL REQUIREMENTS DEFINITION

Category	Description
Authentication and Access Control	University email verification system, role-based access control, token-based authentication.
Event Management	Users can create, update, delete academic events; topic categorization; location services integration; attendance tracking.
Content Management	Faculty can create podcasts and seminars; students need faculty supervision; outsiders can attend but not create content; admin approval required.
User Interaction	Commenting system with moderation, content rating, interest marking, personalized event recommendations, event notifications.
Data Handling	Secure storage of event details, real-time event registration processing, data synchronization across universities.

Table 10 FUNCTIONAL REQUIREMENTS DEFINITION

2.2.1.2. Non-functional

TABLE XI. NON-FUNCTIONAL REQUIREMENTS DEFINITION

Category	Description
Performance & Scalability	Support for 1,000+ concurrent users, 95% of queries under 200ms, optimized database performance, load balancing.
Security & Compliance	MFA for sensitive actions, AES-256 encryption, API rate limiting, GDPR compliance.
Usability & Accessibility	WCAG 2.1 compliance, responsive UI for multiple devices, minimalistic design.
Reliability & Availability	99.9% uptime, automatic failover mechanisms, regular data backups.

Table 11 NON-FUNCTIONAL REQUIREMENTS DEFINITION

2.2.1.3. System & User

TABLE XII. SYSTEM & USER REQUIREMENTS

Category	Description
Technical Requirements	Web-based platform, React.js frontend, Node.js backend, MongoDB/Firebase/AWS database, OAuth 2.0 authentication.
Hardware & Software	Cloud hosting (AWS/Google Cloud/Azure), CI/CD pipeline, GitHub for version control.
User Expectations	Fast, intuitive experience, categorized event listings, university system integration, secure authentication.

Table 12 SYSTEM & USER REQUIREMENTS DEFINITION

2.2.1.4. Business

The system is designed to enhance academic collaboration by creating a centralized platform for discovering and managing academic events across Berlin universities. It aims to increase accessibility to relevant seminars, conferences, and research discussions while providing a user-friendly experience.

A key business goal is to enable personalized recommendations, ensuring users receive event suggestions tailored to their interests and affiliations. The platform also supports university branding by aligning with institutional requirements and ensuring compliance with GDPR regulations.

Future scalability is considered, with potential expansion to integrate external academic conferences and cross-university collaborations

2.2.1.5. Dependencies

The system relies on several dependencies to function effectively. It integrates third-party services such as Google Maps API for location-based event navigation and TransLink API for public transport guidance. University email authentication is essential for verifying user identities and enforcing access control.

Infrastructure dependencies include cloud hosting solutions like AWS, Firebase, or Google Cloud, ensuring scalability, reliability, and data storage. Secure database storage and regular backup services are implemented to prevent data loss.

Operational dependencies involve administrative workflows where university-appointed moderators must approve events before publication. Collaboration with university IT departments is required to ensure smooth authentication processes and secure data exchange across institutions.

2.2.2. Development Methodologies

2.2.2.1. User Behaviors - BDD

User Stories:

"As a student, I want to search for upcoming academic events, so that I can find relevant talks easily."

"As a faculty member, I want to create and manage events, so that I can organize academic discussions."

"As an administrator, I want to approve event submissions, so that only relevant and verified content is published."

Given-When-Then Scenarios:

Event Creation:

- Given a faculty member is logged in,
- When they submit an event,
- Then the event should be listed after approval.

User Registration:

- Given a user with a university email,
- When they attempt to register,
- Then they should receive a verification email.

Behavior Validation Criteria:

- Ensure role-based access restrictions work as expected.
- Validate search results match query filters.
- Verify event approvals require admin validation.

2.2.2.2. Feature List & Prioritization - FDD

TABEL XIII. FEATURE LIST & PRIORITIZATION

Feature	Priority	Dependencies
User Authentication	High	University Email Verification
Event Creation	High	Authentication, Admin Approval
Attendance Tracking	Medium	Authentication
Comment System	Low	Event Creation

Table 13 FEATURE LIST & PRIORITIZATION

2.2.2.3. Feature Development Sequence:

Authentication and Role Management

Event Creation & Approval

User Engagement Features (Comments, Interests, Ratings)

Attendance Tracking and Metrics Collection

2.2.2.4. Feature Completion Criteria:

Authentication allows access to different user roles.

Event creation workflow follows proper approval mechanisms.

Users can track attendance and interact with content.

Performance metrics provide insight into user engagement.

2.2.2.5. Test Specifications TDD:

Define tests before implementing features.

Validate input constraints (e.g., invalid email rejection).

Ensure performance benchmarks are met.

Implement automated test suites integrated into CI/CD pipelines.

TABLE XIV. INITIAL TEST CASES

Test ID	Description	Expected Outcome
T-001	Register with a valid university email	Registration succeeds
T-002	Register with an invalid email	Error message displayed
T-003	Create an event without admin approval	Event remains in pending state
T-004	Search for an event using filters	Relevant results returned

Table 14 INITIAL TEST CASES

2.2.2.6. Test Coverage Requirements - TDD:

Unit Tests: 90% coverage for core functions.

Integration Tests: 85% coverage for critical workflows.

End-to-End Tests: 80% coverage of user actions.

Performance Testing: Load testing to ensure scalability under peak conditions.

2.2.3. Constraints

2.2.3.1. Technological

The system should use **secure authentication mechanisms**.

API limits should be monitored and optimized.

The platform must be **compatible with modern browsers and mobile devices**.

2.2.3.2. Time

The system must be **ready for deployment within 6 months**.

Key milestones should be **defined and strictly followed** to prevent delays.

2.2.3.3. Legal

The system must comply with **GDPR and university data policies**.

All stored user data must be **encrypted and anonymized where necessary**.

2.2.4. Acceptance criteria – BDD & TDD

2.2.4.1. Behavioral benchmarks

Users must be able to register and authenticate within **30 seconds**.

Event creation should have an approval turnaround of **max 12 hours**.

Users should be able to search for events with **accuracy above 95%.**

2.2.4.2. Performance benchmarks

Search queries should return results in **under 200ms**.

User sessions should remain active for at **least 2 hours.**

The platform should **handle up to 1,000 concurrent** users without performance degradation.

2.2.4.3. Testability

Automated test suites should cover **at least 90%** of the codebase.

Continuous Integration (CI) should ensure **all tests pass before merging**.

Regression testing should be conducted before every major release.

3. ESTABLISH

3.1. Architecture And Interface Desing

3.1.1. Prototype

3.1.1.1. Product Backlog – FDD & BDD

Epic: Authentication & User Management

High Priority

- As a student, I want to login with my university email credentials so that I can access the platform securely (Done)
- As an admin, I want to assign faculty member status to eligible users so that they can access appropriate permissions (Done)
- As an outsider, I want to register without a university email so that I can access public features (Done)

Medium Priority

- As a user, I want to update my profile information so that my details are current (In Progress)

- As an admin, I want to view and manage users within my faculty so that I can maintain proper oversight (In Progress)

Epic: Event Management

High Priority

- As a faculty member, I want to create and schedule events/seminars so that students can attend them (Done)
- As a student, I want to view both in-person and online events so that I can choose how and what to attend (Done)
- As an admin, I want to approve or decline event requests so that I can maintain quality control (Done)
- As a user, I want to mark my interest in events so that I can track them later (Done)

Medium Priority

- As a faculty member, I want to create "floating" events without assigned topics so that I can organize independent activities. (In Progress)
- As an admin, I want to assign floating events to topics so that content remains organized (In Progress)
- As a user, I want to view event directions or online links so that I can easily attend (Done)

Epic: Podcast Management

High Priority

- As a student, I want to submit podcast creation requests so that I can share academic content (Done)
- As a faculty member, I want to approve/decline student podcast requests so that I can ensure quality (In Progress)
- As an admin, I want to give final approval for podcasts so that I can maintain platform standards (Done)
- As a user, I want to listen to approved podcasts so that I can access educational content (Done)

Medium Priority

- As a faculty member, I want to create podcasts independently so that I can share knowledge directly (In Progress)
- As a user, I want to browse podcasts by topic so that I can find relevant content (Done)

Epic: Content Interaction

High Priority

- As a registered user, I want to comment on features so that I can engage in discussions (Done)
- As a user, I want to edit or delete my own comments so that I can manage my contributions (In Progress)

- As an admin, I want to moderate comments so that I can maintain community standards (In Progress)

Medium Priority

- As a user, I want to express interest in specific content so that I can save it for later (Done)
- As a registered user, I want to view my interaction history so that I can track my activity (Done)

Epic: Content Discovery

High Priority

- As a user, I want to view active events so that I can join ongoing broadcasts
- As a user, I want to access my subscribed topics so that I can follow relevant content
- As a user, I want to see popular events so that I can discover trending content

Medium Priority

- As a user, I want to receive personalized recommendations so that I can discover relevant content (Done)
- As a user, I want to manage my wishlist so that I can track interesting content (Done)
- As a user, I want to browse content by topics so that I can find specific areas of interest (Done)

Epic: Administrative Tools

High Priority

- As an admin, I want to create and manage topics so that content remains organized (Done)
- As an admin, I want to view analytics on event attendance so that I can track engagement (In Progress)
- As an admin, I want to manage content within my faculty so that I can maintain oversight (Done)

Medium Priority

- As an admin, I want to generate reports on platform usage so that I can assess engagement (In Progress)
- As an admin, I want to manage faculty-specific settings so that I can customize features (In Progress)

TABLE XV. KANBAN

Backlog	To Do	In Progress	Done
Enhancements	**Content Discovery**	**User Management**	**Authentication**
Multi-language support	View Active Events	Update Profile Information	Login with university email
Activity Analytics	Access Subscribed Topics	Admin User Management	Outsider registration
Potential Features		**Event Management**	**Event Features**
Export Event Data		Create floating events	Create/schedule events
Batch Event Creation		Assign floating events to topics	View in-person/online events
Comment moderation			Approve/decline events
			Mark interest in events
			View event directions/links
		Podcast Management	**Podcast Features**
		Faculty member approval/decline	Submit podcast creation request
		Independent Podcast creation (faculty)	Admin final approval
			Listen to podcasts
			Browse podcasts by topic
		Content Interaction	**User Interactions**
		Edit/delete comments	Comment on features
			Express interest in content
		Administrative	**Content Management**
		Manage faculty settings	Create/manage topics (admins)
			Personalized recommendations
			Manage Wishlist
			Browse by topics

Table 15 KANBAN BOARD

3.1.1.2. Design Mockups

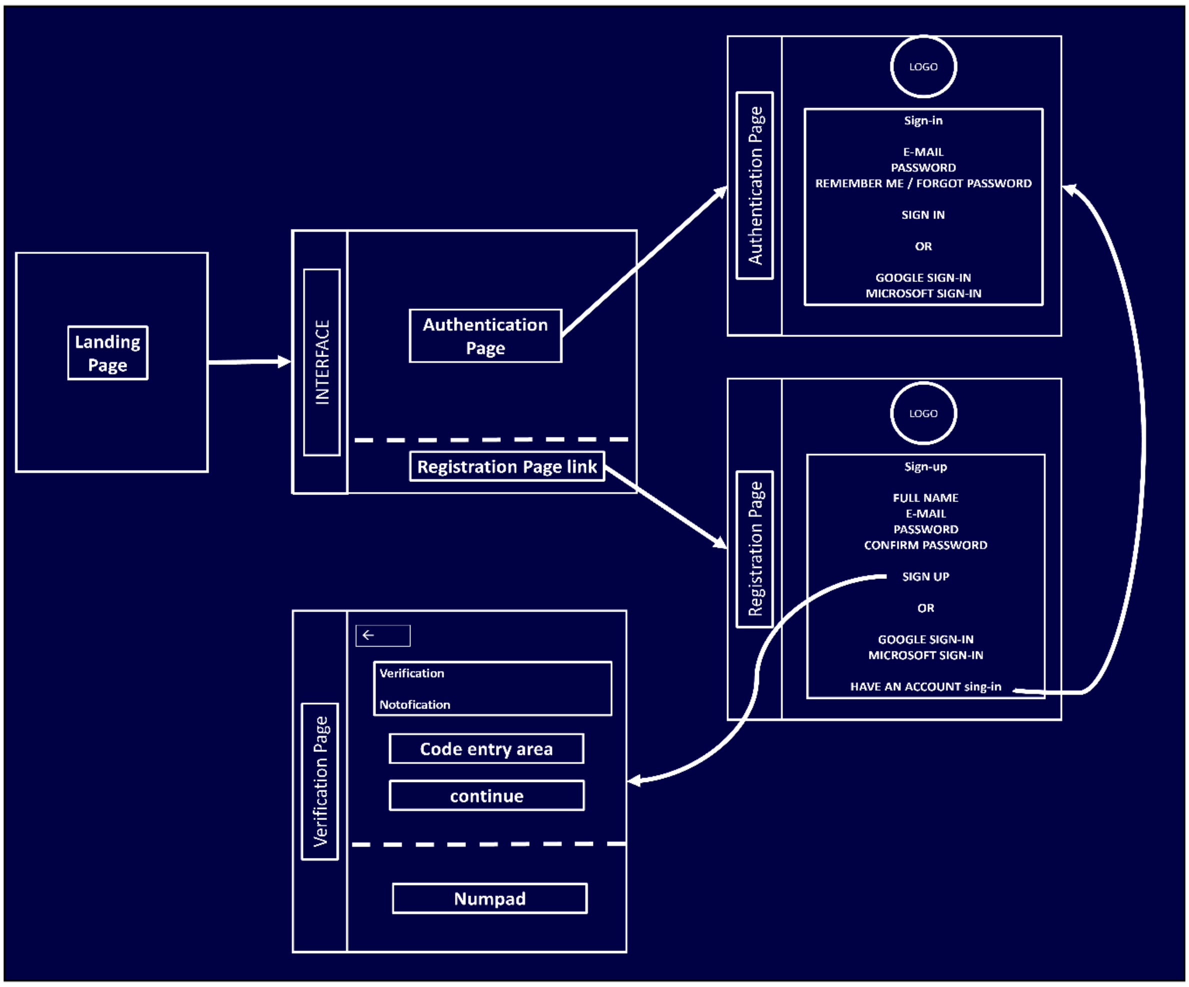

Figure 7 Sign-in / Sign-up Page Designs

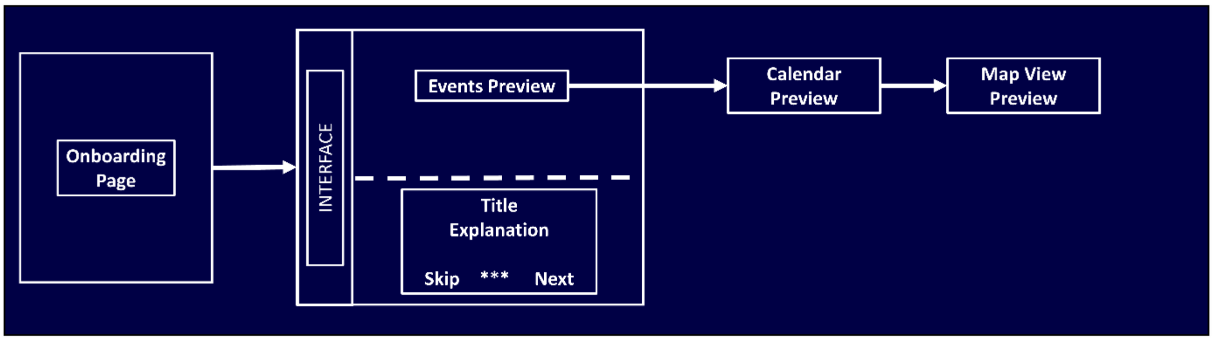

Figure 6 Onboarding Design

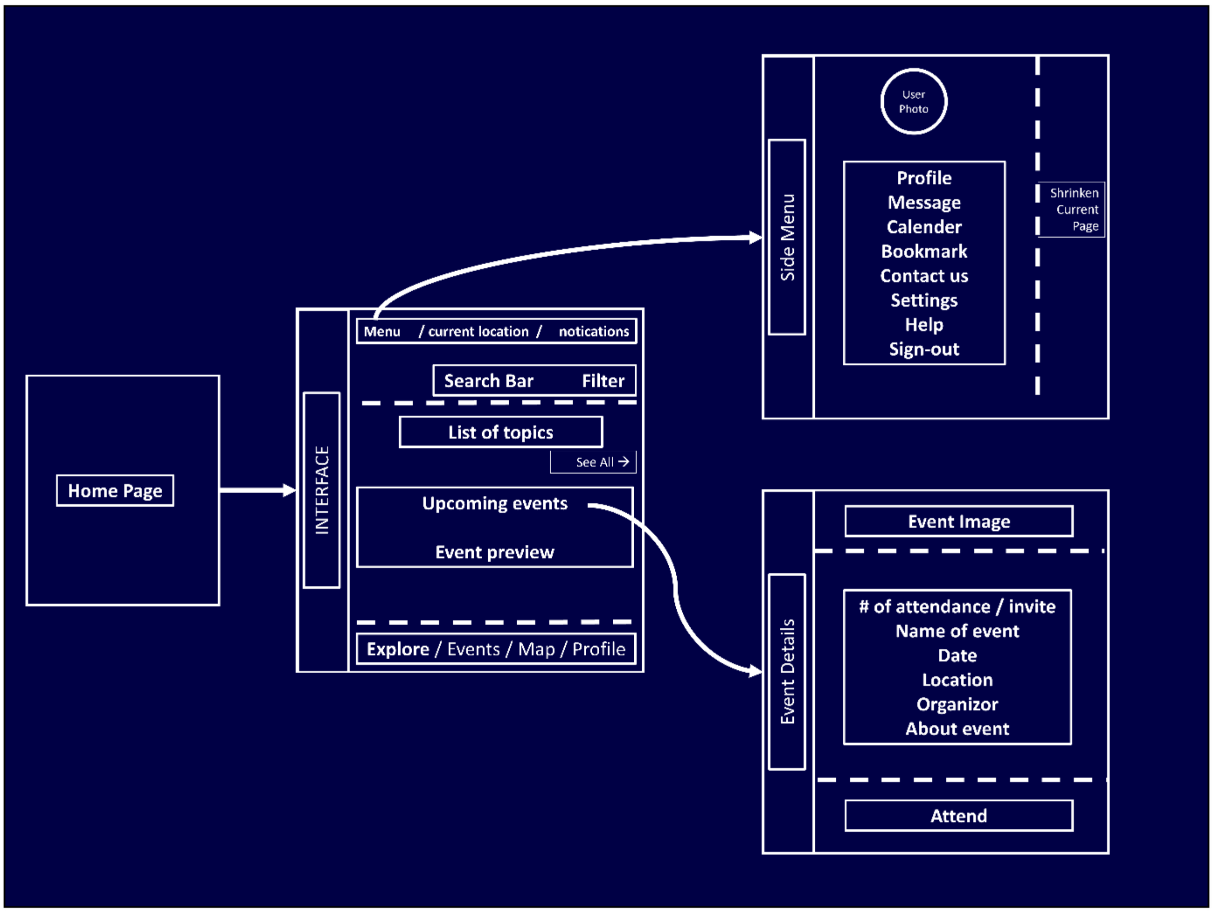

Figure 8 Main Page Design

3.1.1.3. Dev-Ops Planning

1. **System Architecture Layers & Relationships**

Presentation Layer ⟶ Business Logic Layer

- Direct dependency relationship
- Interactions:
  - User interface components to backend services
  - Form submissions and validation
  - Real-time updates and state management
- Data Flow:
  - UI events ⟶ API calls ⟶ Business logic processing
  - Business logic responses ⟶ UI updates

Business Logic Layer ⟶ Data Access Layer

- Bidirectional dependency
- Interactions:
  - Data retrieval and storage operations

- Query processing and optimization
- Transaction management

- Data Flow:
  - Business logic requests → Data queries
  - Data results → Business logic processing

Data Access Layer → External Services Layer

- Integration dependency
- Interactions:
  - Maps and transport API integrations
  - Authentication service connections
  - Email service communications
- Data Flow:
  - Data requests → External API calls
  - External service responses → Data processing

2. **Design Considerations & Reasons**
   - Layered Architecture Choice:
     - Separation of concerns for easier maintenance
     - Independent scaling of components
     - Clear responsibility boundaries
     - Facilitated testing and debugging
   - Performance Metrics Integration:
     - Ensures meeting SLA requirements (1000+ users, 200ms response)
     - Facilitates capacity planning
     - Enables performance optimization
   - Security Features Distribution:
     - Multi-layer security approach
     - Data protection at all levels
     - GDPR compliance maintenance
3. **Infrastructure Management**
   - Caching Strategy:
     - Reduces database load

  - Improves response times
  - Handles concurrent users efficiently
- Monitoring and Logging:
  - Proactive issue detection
  - Troubleshooting facilitation
  - Ensures 99.9% uptime requirement
- Scalability Considerations:
  - Supports increasing user base
  - Maintains performance under load
  - Enables geographic distribution
- Error Handling:
  - Ensures graceful failure handling
  - Maintains user experience
  - Enables system recovery

4. **Key Benefits:**
   - The layered approach ensures:
     - High maintainability
     - Scalable architecture
     - Secure operations
     - Efficient performance
     - Reliable service delivery
     - Clear system organization
     - Easy troubleshooting
     - Future extensibility

3.1.2. System Architecture

3.1.2.1. Component Design

3.1.2.1.1. FSM model

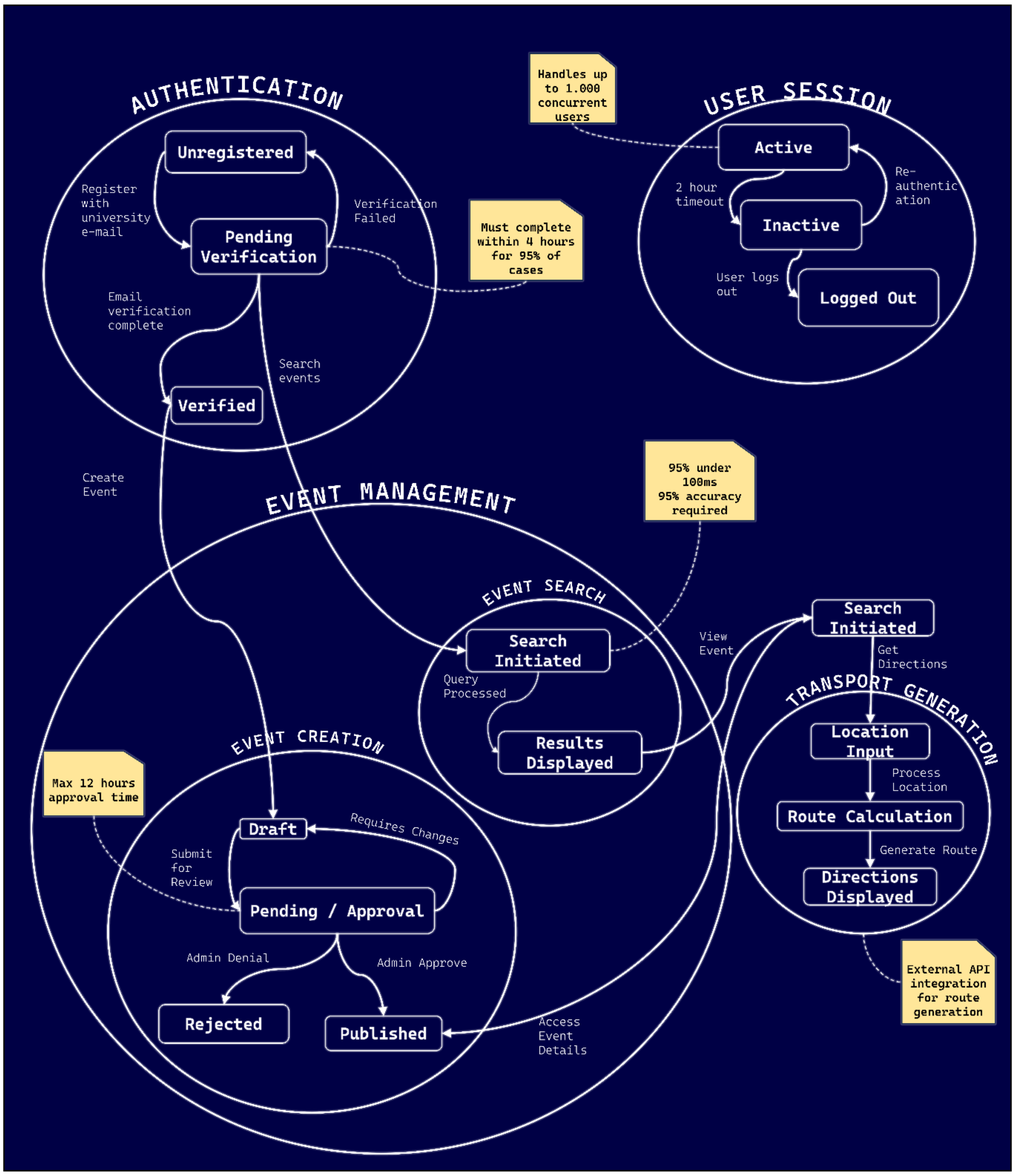

Figure 9 Finite State Machine Diagram

3.1.2.1.2. Layer Relationships - RDD

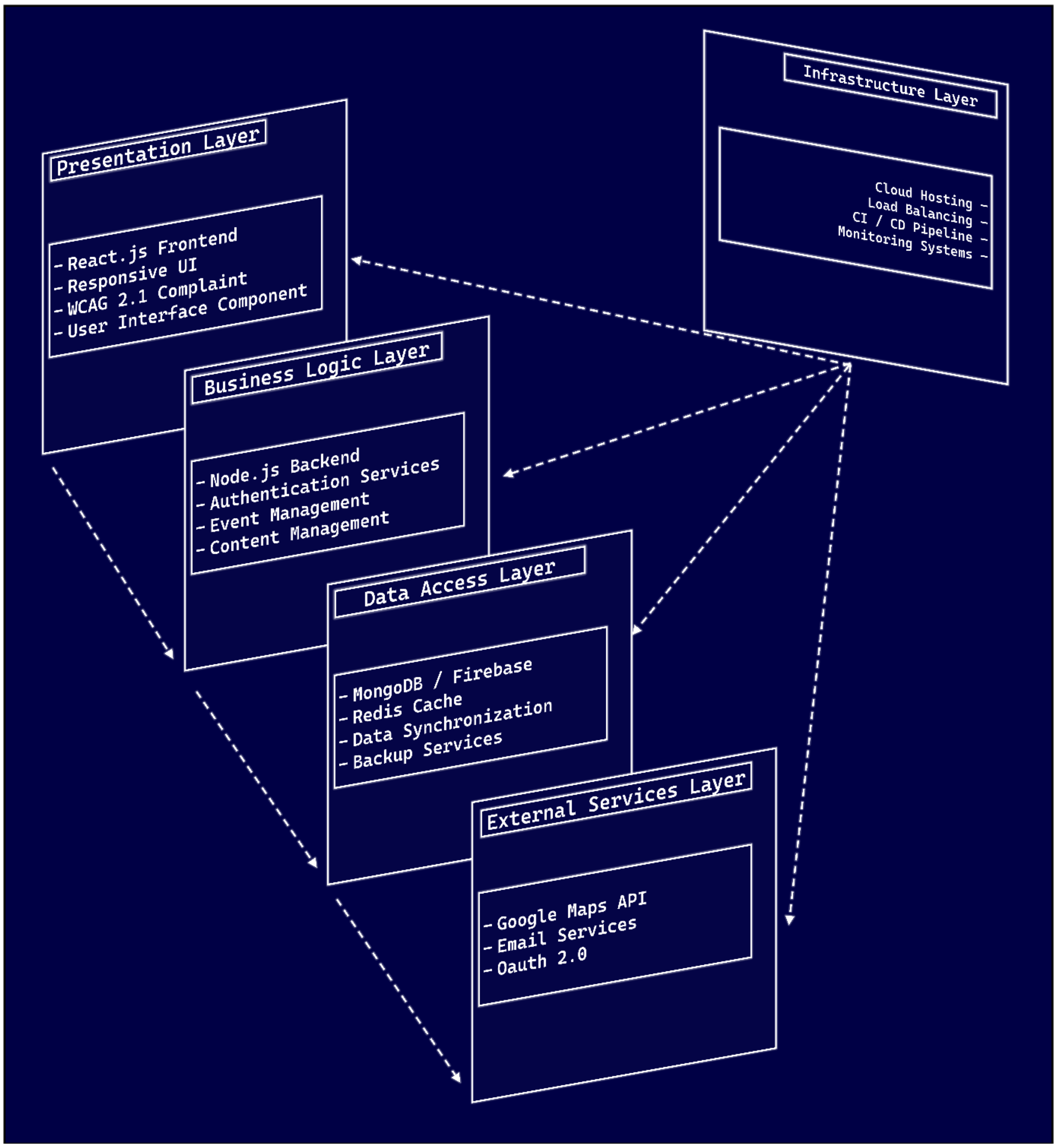

Figure 10 Layer Relationships Diagram

1. **Presentation Layer → Business Logic Layer**
    - **Relationship Type:** Direct dependency
    - **Interactions:**
        - User interface components make requests to backend services
        - Form submissions and data validation
        - Real-time updates and state management

- **Data Flow:**
  - UI events → API calls → Business logic processing
  - Business logic responses → UI updates

2. **Business Logic Layer → Data Access Layer**
   - **Relationship Type:** Bidirectional dependency
   - **Interactions:**
     - Data retrieval and storage operations
     - Query processing and optimization
     - Transaction management
   - **Data Flow:**
     - Business logic requests → Data queries
     - Data results → Business logic processing
3. **Data Access Layer → External Services Layer**
   - **Relationship Type:** Integration dependency
   - **Interactions:**
     - API integrations for maps and transport
     - Authentication service connections
     - Email service communications
   - **Data Flow:**
     - Data requests → External API calls
     - External service responses → Data processing
4. **Infrastructure Layer ↔ All Other Layers**
   - **Relationship Type:** Supporting infrastructure
   - **Interactions:**
     - Hosting and deployment
     - Load balancing and scaling
     - Monitoring and logging
   - **Data Flow:**
     - Cross-cutting concerns across all layers

Design Considerations and Reasons:

1. **Layered Architecture Choice**

- **Consideration:** Separation of concerns
- **Reason:**
    - Makes system easier to maintain and modify
    - Allows independent scaling of different components
    - Enables clear responsibility boundaries
    - Facilitates testing and debugging

2. **Overlapping Boxes Design**
    - **Consideration**: Layer interaction visualization
    - **Reason:**
- Shows real-world interdependencies
- Illustrates cross-cutting concerns
- Demonstrates how components interact across layers

3. **Infrastructure Layer Placement**
    - **Consideration:** Supporting all other layers
    - **Reason:**
        - Provides foundation for all system operations
        - Enables system-wide monitoring and management
        - Facilitates scalability and reliability

4. **Performance Metrics Integration**
    - **Consideration:** System requirements monitoring
    - **Reason:**
        - Ensures meeting SLA requirements (1000+ users, 200ms response)
        - Facilitates capacity planning
        - Enables performance optimization

5. **Security Features Distribution**
    - **Consideration:** Multi-layer security approach
    - **Reason:**
        - Implements defense in depth
        - Ensures data protection at all levels
        - Maintains compliance requirements (GDPR)

6. **External Services Integration**
    - **Consideration:** Third-party service management
    - **Reason:**
        - Leverages existing solutions
        - Reduces development time
        - Provides specialized functionality
7. **Caching Strategy**
    - **Consideration:** Performance optimization
    - **Reason:**
        - Reduces database load
        - Improves response times
        - Handles concurrent users efficiently
8. **Monitoring and Logging**
    - **Consideration:** System health tracking
    - **Reason:**
        - Enables proactive issue detection
        - Facilitates troubleshooting
        - Ensures 99.9% uptime requirement
9. **Scalability Considerations**
    - **Consideration:** Growth and load handling
    - **Reason:**
        - Supports increasing user base
        - Maintains performance under load
        - Enables geographic distribution
10. **Error Handling**
    - **Consideration:** System resilience
    - **Reason:**
        - Ensures graceful failure handling
        - Maintains user experience
        - Enables system recovery

This layered approach with its considerations ensures:

- High maintainability
- Scalable architecture
- Secure operations
- Efficient performance
- Reliable service delivery
- Clear system organization
- Easy troubleshooting
- Future extensibility

3.1.2.2. UML Class Diagrams

3.1.2.2.1. Component Diagram

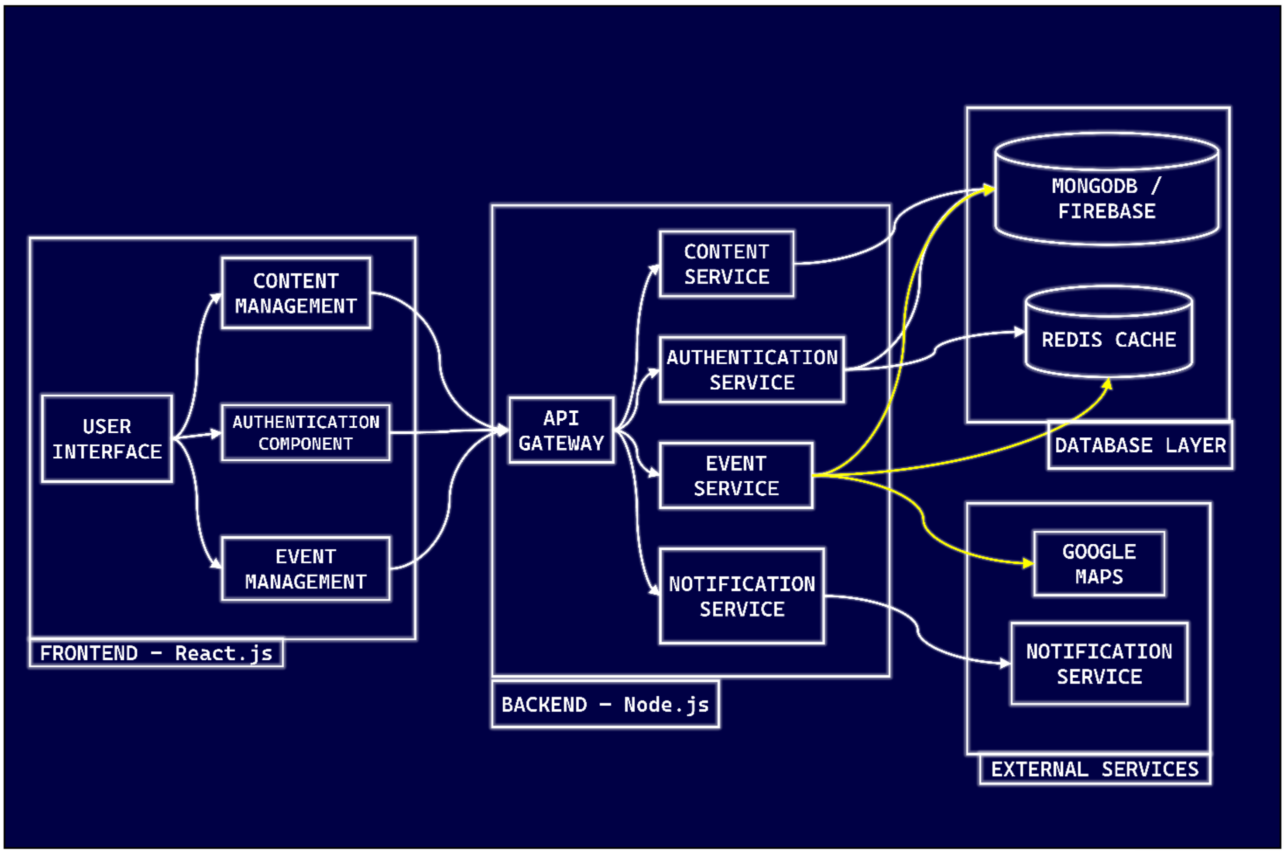

Figure 11 Component Diagram

- **Frontend Layer (React.js):**
  1. User Interface: Main UI components
  2. Authentication Component: Handles user authentication
  3. Event Management: Event-related UI components
  4. Content Management: Content handling components
- **Backend Layer (Node.js):**

1. API Gateway: Central entry point for all requests
2. Services:
    1. Authentication Service: Handles user verification and access
    2. Event Service: Manages event operations
    3. Content Service: Handles content operations
    4. Notification Service: Manages system notifications

- **Database Layer:**
    1. MongoDB/Firebase: Primary data storage
    2. Redis Cache: Performance optimization through caching
- **External Services:**
    1. Google Maps API: Location services
    2. TransLink API: Transport information
    3. Email Service: Communication service
- **Key Interactions:**
    1. Frontend components → API Gateway
    2. API Gateway → Various backend services
    3. Services → Database and Cache
    4. Services → External APIs
- **Data Flow:**
    1. UI components make requests through the API Gateway
    2. Gateway routes requests to appropriate services
    3. Services interact with database and external APIs
    4. Results flow back through the same path

This architecture follows key requirements:

- **Scalability** (through service separation)
- **Performance** (through caching)
- **Security** (through authentication layer)
- **Modularity** (through component separation)
- **Integration** capability (through external services)

3.1.2.2.2. Class Diagram

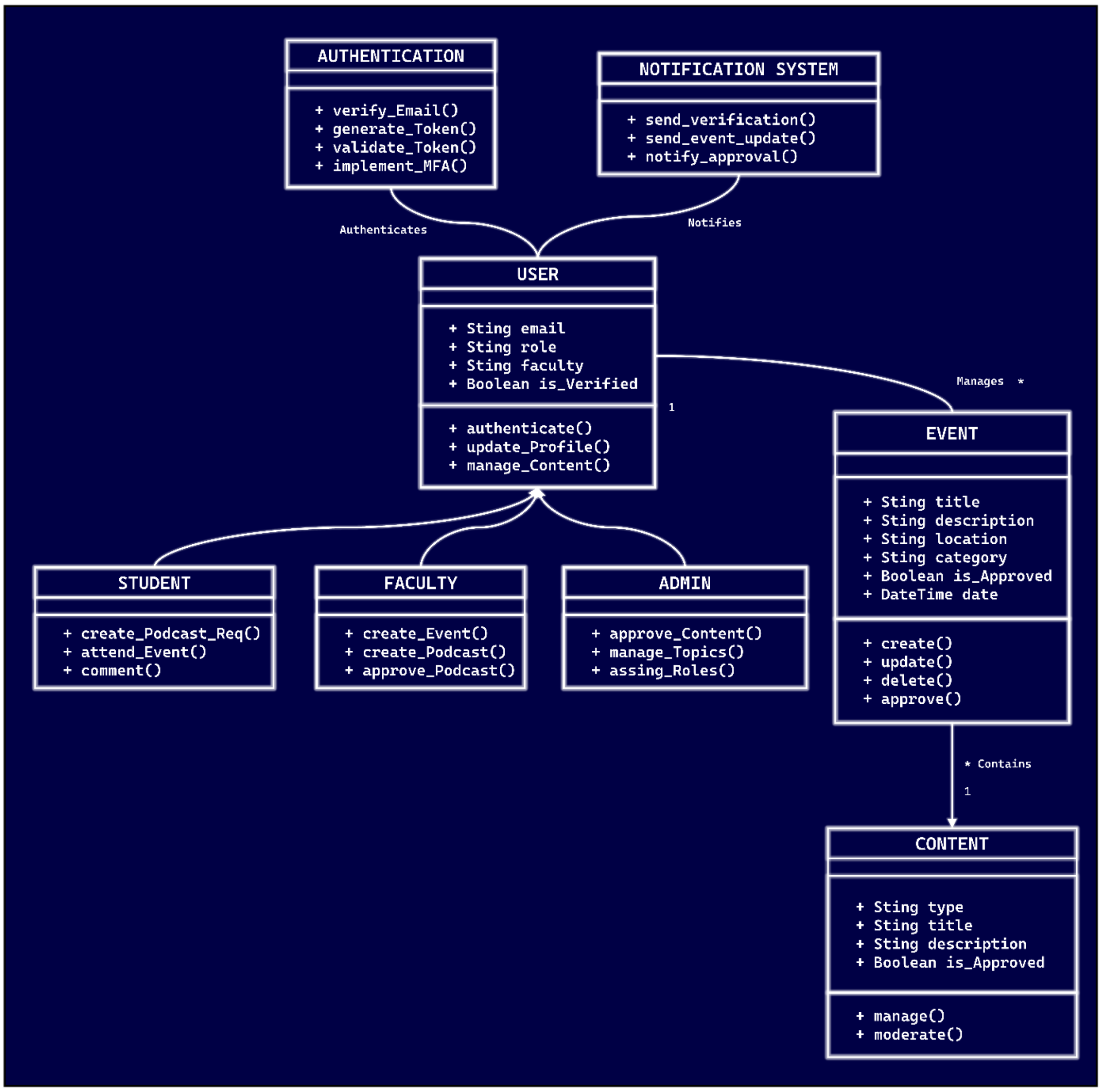

Figure 12 Class Diagram

- **User Class Hierarchy:**

  Base User class with common attributes:

  `email, role, faculty, is_Verified`

  Basic operations: `authenticate(), update_Profile(), manage_Content()`

  Three specialized user types inherit from base User:

  Student: Can request podcasts, attend events, and comment

  Faculty: Can create events/podcasts and approve podcast requests

  Admin: Has highest privileges - approving content, managing topics, assigning roles

- **Content Management Classes:**

  Event class:

  Attributes: title, description, date, location, category, approval status

  Operations: CRUD operations (create, update, delete) and approve

  Content class:

  Represents general content (podcasts, seminars)

  Attributes: type, title, description, approval status

  Operations: `manage()` and `moderate()`

- **Support Classes:**

  Authentication: Handles email verification, token management, and MFA

  Notification System: Manages all system notifications including verifications and updates

- **Key Relationships:**

  User "1 to many" Events: One user can manage multiple events

  Event "many to 1" Content: Events contain content items

  Authentication -- User: Authentication service connects to users

  Notification System -- User: Notification service connects to users

3.1.2.2.3. ERE Model

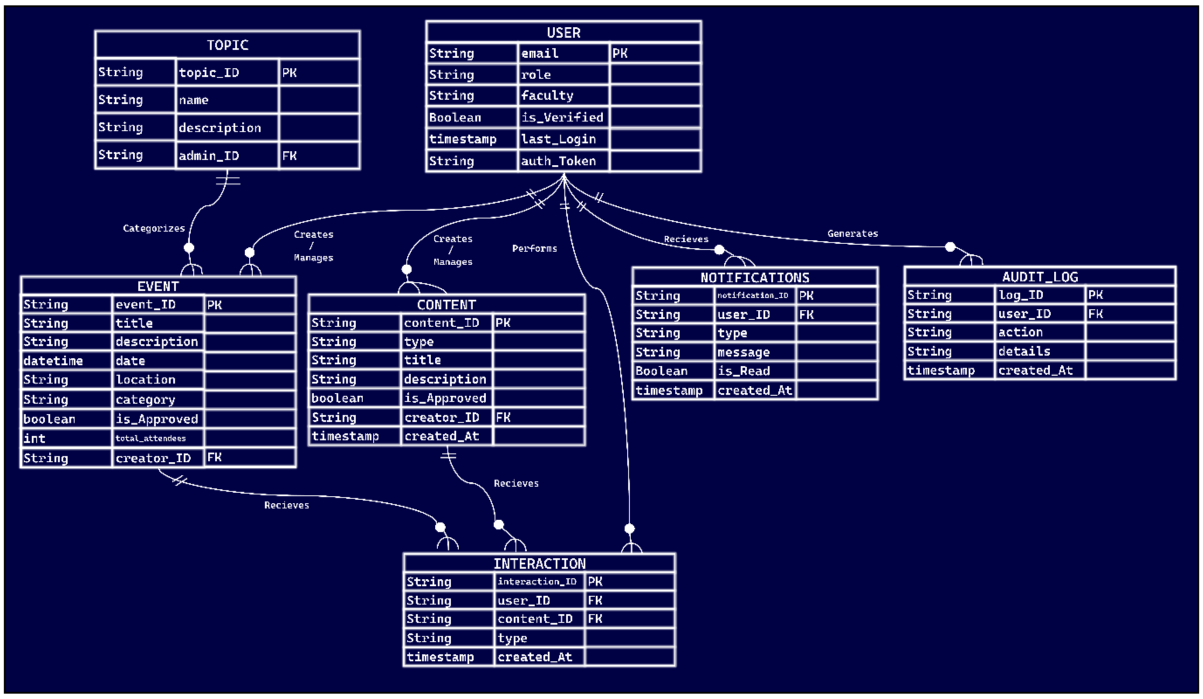

Figure 13 ERE Diagram

1. **Core Entities: USER Entity:**
    1. Primary key: email
    2. Attributes: role, faculty, verification status
    3. Key events:
        1. User registration
        2. Email verification
        3. Role assignment
        4. Authentication
2. **EVENT Entity:**
    1. Primary key: event_ID
    2. Attributes: title, description, date, location
    3. Key events:
        1. Event creation
        2. Approval status change
        3. Attendee updates
        4. Location updates
3. **CONTENT Entity:**

   1. Primary key: content_ID
   2. Attributes: type, title, approval status
   3. Key events:
      1. Content creation
      2. Content approval
      3. Content updates
      4. Content archival
4. **Supporting Entities: INTERACTION Entity:**
   1. Records user interactions with events/content
   2. Tracks:
      1. Comments
      2. Ratings
      3. Attendance
      4. Interest markers
5. **TOPIC Entity:**
   1. Organizes events and content
   2. Managed by admins
   3. Enables categorization
6. **NOTIFICATION Entity:**
   1. Handles system communications
   2. Types:
      1. Event updates
      2. Content approvals
      3. System notifications
      4. Reminders
7. **AUDIT_LOG Entity:**
   1. Tracks system actions
   2. Security monitoring
   3. Compliance recording
8. **Key Relationships: User Relationships:**
   1. User → Event (1:Many): Users create/manage multiple events

    2. User → Content (1:Many): Users create/manage various content
    3. User → Interaction (1:Many): Users interact with events/content
    4. User → Notification (1:Many): Users receive notifications

9. **Event Relationships:**
    1. Event → Topic (Many:1): Events belong to topics
    2. Event → Interaction (1:Many): Events receive interactions
10. **Content Relationships:**
    1. Content → Topic (Many:1): Content categorization
    2. Content → Interaction (1:Many): Content receives interactions
11. **Event Triggers: User Events:**
    1. Registration triggers email verification
    2. Role changes trigger permission updates
    3. Actions trigger audit logging
12. **Content Events:**
    1. Creation triggers approval workflow
    2. Updates trigger notifications
    3. Interactions trigger recommendations
13. **System Events:**
    1. Scheduled event reminders
    2. Automatic content archival
    3. Performance monitoring
14. **Data Flow:**
    1. User actions generate events
    2. Events trigger notifications
    3. Interactions update recommendations
    4. All actions logged for audit

This ERE diagram supports:

- GDPR compliance through audit logging
- Scalable event management
- Complex user interactions
- Notification system

- Role-based access control
- Content moderation workflow

3.1.2.3. Technology Stack

3.1.2.3.1. Backend

Node.js with Express.js framework

- REST API development
- WebSocket for real-time features
- TypeScript for type safety

Services:

- User Authentication Service
- Event Management Service
- Content Delivery Service
- File Upload Service

Database Architecture:

- Primary Database: MongoDB
    - Stores user data, events, content
    - Flexible schema for future scalability
- Cache Layer: Redis
    - Session management
    - Temporary data caching
    - Real-time data handling

3.1.2.3.2. Frontend

React Native for cross-platform development

JavaScript/TypeScript for core development

Platform-specific code: Swift (iOS) and Kotlin (Android)

Core Components:

- Navigation (React Navigation)
- State Management (Redux/Context API)
- Authentication UI
- Event/Content Management UI
- Push Notification Handling

3.1.2.3.3. API communication

RESTful APIs for main services

WebSocket for real-time features

External API Integration:

- Google Maps API
- MongoDB API
- Payment gateways (if needed)

Development Requirements:

- Languages:
    - JavaScript/TypeScript (Frontend & Backend)
    - Swift (iOS native modules)
    - Kotlin (Android native modules)
- Development Tools:
    - Xcode (iOS)
    - Android Studio
    - VS Code
    - Git for version control

3.1.2.4. Data Flow - DDD

3.1.2.4.1. API layer

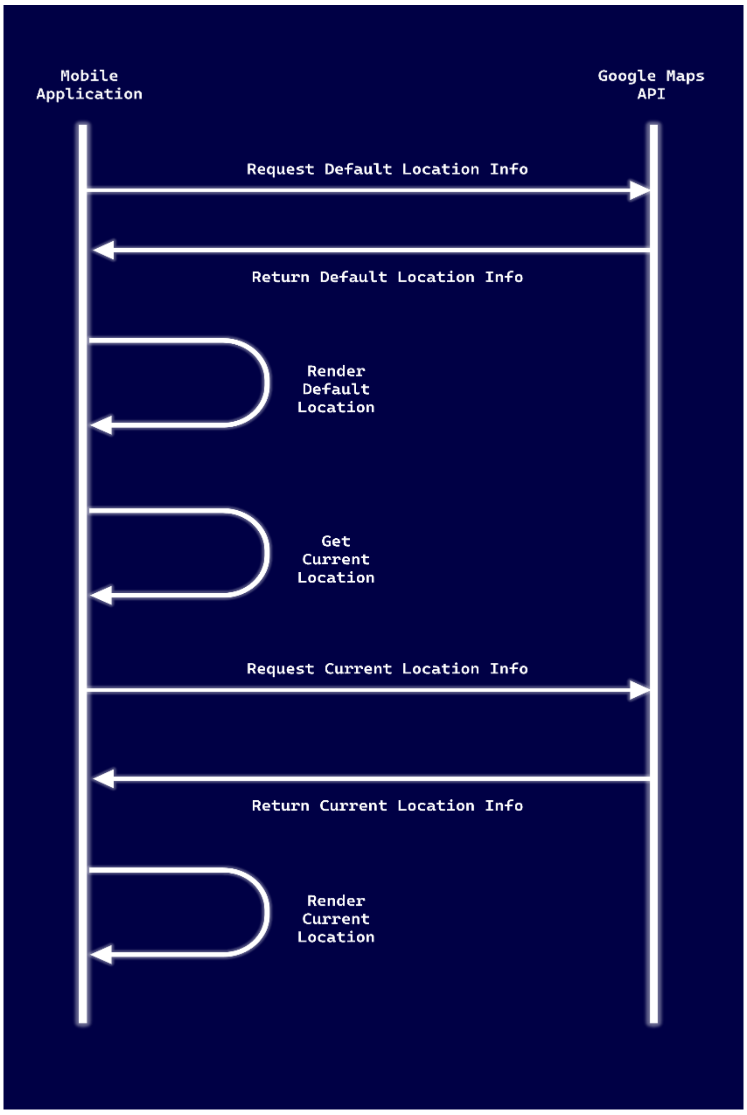
Mobile Application
Google Maps API
Request Default Location Info
Return Default Location Info
Render Default Location
Get Current Location
Request Current Location Info
Return Current Location Info
Render Current Location

Figure 14 API Layer

3.1.3. Interface Desing

3.1.3.1. Usability Goals - DDD

Intuitive Navigation:

The platform should allow users to easily create, find, and manage events with minimal effort.

A clean, organized UI ensures users understand the FSM states and actions available at each step.

Fast & Responsive Interactions:

Event search results should load in under 100ms for 95% of cases.

Email verification should complete within 4 hours for 95% of cases to optimize account activation.

Scalability & Performance:

The system must handle up to 1,000 concurrent users without performance degradation.

Real-time status updates for event approvals and search results should be handled efficiently.

Accessibility & Mobile-Friendliness:

The platform should be WCAG 2.1 compliant to accommodate users with disabilities.

A responsive design ensures usability on both desktop and mobile devices.

3.1.3.2. Feedback Mechanisms

Real-Time Status Updates:

Users receive live feedback during event creation and approval.

If an event is rejected, the system provides specific reasons and required changes.

Error Handling & Notifications:

Friendly error messages guide users when verification fails or session timeouts occur.

Toast notifications confirm successful actions (e.g., "Event Published Successfully!").

Search Optimization Feedback:

When an event search returns no results, the system suggests related events or alternative queries.

Users receive feedback on expected response times if a search query is complex.

Session & Authentication Feedback:

Clear messaging when a session expires (e.g., “You have been logged out due to inactivity”).

Re-authentication prompts ensure a smooth transition without losing user progress.

3.1.3.3. Prototype Demonstration

To provide a visual and interactive representation of the Unified Academic Event Platform, we have developed the prototype using Figma. This prototype showcases the core user flows, interface design, and interaction elements that users will experience in the final implementation.

The prototype includes:

User Authentication: Login and registration process with university email verification.

Event Management: Creating, editing, and browsing academic events.

Podcast Hosting: Faculty-managed podcast creation and streaming.

Navigation & Search: Smart filtering and recommendation features.

Transport Integration: Automated generation of location-based transport links.

You can access and interact with the Figma prototype using the following link: Shayan Rhimi: https://www.figma.com/design/jmz6QWHtIo7u27k3tftEM3/EventHubASD?node...