

Orgifi

Design Document

Team 38

Client: Hunter Barton

Advisor: Mai Zheng

Team Members:

Hunter Barton - Project Manager and UX/UI developer

Perry Ports - Backend Developer

Adin Huric - Backend Developer

Dino Huric - Frontend Developer

Tabé Ekpombang - Backend Developer

Hongwei Wang - Communicator & Frontend Developer

Mohammed Abdalgader - Frontend & WebSocket developer

Team Email: sdmay25-38@iastate.edu

Team Website: <https://sdmay25-38.sd.ece.iastate.edu>

Revised: 05/03/2025

Executive Summary

Orgifi is a user-friendly web application created to help people find, join, and manage organizations easily in one convenient location. The application simplifies the process of discovering groups of interest, becoming an active member, and effectively managing organizational tasks, making community involvement more accessible and engaging. Finding a Community is a huge problem for people who move to new places as they usually do not know where to start at finding a community. Orgifi's focus is to make this transition to a new place easier as we will provide them with a means to easily find and join a community.

Design requirements.

Orgifi's key requirements include the following:

1. **Unified discovery & onboarding** – search for clubs and organizations by geo location or interest, request membership in a few clicks.
2. **Role-based access control (RBAC)** – three tiers (Admin, Executive, Member) with strict permission boundaries.
3. **Real-time collaboration** – group chat and direct messages for easy communication amongst members.
4. **Integrated calendaring** – shared event creation for group events.

Our system is built as a three-part web platform that keeps what users see, what the server decides, and where the data lives clearly separated. People browse or search for clubs, submit a join request, and once approved, they can chat in real time and follow a shared calendar. Leaders see extra controls for recruiting and scheduling, while members only see features they need. This separation of duties makes the interface simple for end-users and the codebase easier to grow or scale.

Technologies used.

- **Frontend:** React 18 with Next.js for fast, easily viewable pages; TailwindCSS for styling; Redux for predictable state.
- **Backend:** Node.js and Express, written in TypeScript for type safety; Socket.IO handles live chat and calendar updates.
- **Data & identity:** MongoDB Atlas stores clubs, members, and messages; Firebase Authentication manages secure log-ins; Firebase Storage hosts user images under access rules.
- **DevOps:** A monorepo houses all services; GitHub Actions run tests and build Docker containers, which deploy to cloud infrastructure with minimal downtime.

Based on tests we have carried out, Orgifi meets its requirements as it has an easy work flow where its users can easily search for groups based on their interests, join them and use the chat and calendar features with ease, which gives them a sense of community as hoped for.

Some potential implications will include a need to upgrade on the security implications depending on how much Orgifi grows. The next steps to be taken with our project include we'll connect the app to a trusted payment service (e.g., Stripe) so members can pay dues or event fees without leaving the platform.

Learning Summary

Development Standards & Practices Used

Software Practices:

- Agile software development methodologies
- Domain-Driven Design (DDD)
- Microservice architecture
- Code modularity and reusability
- Continuous integration and continuous deployment (CI/CD)
- Unit, integration, and end-to-end testing
- Version control using Git
- Clear documentation and commenting within code
- User experience (UX) best practices

Engineering Standards Considered:

- IEEE 830: Recommended practice for software requirements specifications
- IEEE 1016: Recommended practice for software design descriptions
- IEEE 1028: Standard for software reviews and audits
- IEEE 1061: Standard for Software Quality Metrics Methodology
- ISO/IEC 25010: Software product quality requirements and evaluation (SQuaRE)
- Web Content Accessibility Guidelines (WCAG) 2.1 for accessibility compliance
- Data security standards and best practices, including secure authentication and data encryption

These standards and practices have guided the development process to ensure Orgifi delivers a reliable, maintainable, and user-friendly web application.

Summary of Requirements:

- Users can search for organizations by name or location.
- Users can request to join organizations.
- Organization leaders can approve or deny membership requests.
- Members can send direct messages and participate in group chats.
- Organizations have integrated calendars for event tracking.
- Leaders can create, edit, and manage organizational events.
- Leaders can manage member roles and permissions.
- Leaders can update organization profiles and details.
- Future integration of payment processing for membership dues.

Applicable Courses from the Iowa State University Curriculum

- **COMS 309: Software Development Practices** - Provides practical experience in software development, including design, implementation, and testing of software systems. [OBJ]
- **SE 329: Software Project Management** - Focuses on project management principles, including planning, scheduling, risk management, and quality assurance in software projects. [OBJ]
- **SE 339: Software Architecture and Design** - Covers architectural design principles, design patterns, and the role of architecture in software development. [OBJ]
- **SE 409: Software Requirements Engineering** - Emphasizes the requirements engineering process, including elicitation, analysis, specification, and validation. [OBJ] [OBJ]
- **SE 417: Software Testing** - Introduces software testing principles and techniques, including test models, test design, and software testing tools. [OBJ]
- **COMS 227: Object-Oriented Programming** - Covers object-oriented programming concepts, including classes, inheritance, and polymorphism. [OBJ]
- **COMS 363: Introduction to Database Management Systems** - Introduces database concepts, including data models, relational databases, and SQL. [OBJ]
- **COMS 417: Software Testing** - Provides an introduction to software testing principles and techniques, including regression and system testing. [OBJ]

New Skills/Knowledge acquired that were not taught in courses

- Advanced usage and setup of TypeScript monorepos.
- Practical implementation and management of microservice architecture.
- Extensive hands-on experience with Next.js for server-side rendering.
- Integration and management of Firebase authentication and storage.
- Deployment and management of MongoDB databases in a production environment.
- Real-world experience with Domain-Driven Design (DDD).
- Configuration and maintenance of continuous integration and continuous deployment (CI/CD) pipelines.
- Enhanced skills in UX/UI design principles specifically tailored for web applications.
- Real-time messaging and chat integration in web applications.
- Advanced state management techniques using Redux.

Table of Contents

1. Introduction.....	6
1.1. Problem Statement.....	6
1.2. Intended Users.....	6
2. Requirements, Constraints, and Standards.....	7
2.1. Requirements and Constraints.....	7
2.2. Engineering Standards.....	7
3. Project Plan.....	8
3.1. Project Management / Tracking Procedures.....	8
3.2. Task Decomposition.....	8
3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria.....	8
3.4. Project Timeline/Schedule.....	8
3.5. Risks and Risk Management.....	8
3.6. Personnel Effort Requirements.....	8
3.7. Other Resource Requirements.....	9
4. Design.....	10
4.1. Design Context.....	10
4.1.1. Broader Context.....	10
4.1.2. Prior Work/Solutions.....	10
4.1.3. Technical Complexity.....	10
4.2. Design Exploration.....	10
4.2.1. Design Decisions.....	10
4.2.2. Ideation.....	10
4.2.3. Decision-Making and Trade-Off.....	10
4.3. Final Design.....	11
4.3.1. Overview.....	11
4.3.2. Detailed Design and Visual(s).....	11
4.3.3. Functionality.....	11
4.3.4. Areas of Challenge.....	11
4.4. Technology Considerations.....	11
5. Testing.....	12
5.1. Unit Testing.....	12
5.2. Interface Testing.....	12
5.3. Integration Testing.....	12
5.4. System Testing.....	12
5.5. Regression Testing.....	12
5.6. Acceptance Testing.....	12
5.7. User Testing.....	13
5.8. Other Types of Testing.....	13
5.9. Results.....	13
6. Implementation.....	14

6.1. Design Analysis.....	14
7. Ethics and Professional Responsibility.....	15
7.1. Areas of Professional Responsibility/Code of Ethics.....	15
7.2. Four Principles.....	15
7.3. Virtues.....	15
8. Conclusions.....	16
8.1. Summary of Progress.....	16
8.2. Values Provided.....	16
8.3. Next Steps.....	16
9. References.....	17
10. Appendices.....	18
Appendix 1 – Operation Manual.....	18
Appendix 2 – Alternative/Initial Version of Design.....	18
Appendix 3 – Other considerations.....	18
Appendix 4 – Code.....	18
Appendix 5 – Team Contract.....	18

1. Introduction

1.1. Problem Statement

Today, individuals often struggle to find, join, and actively participate in local organizations due to fragmented communication channels, limited information, and organizational inefficiencies. People interested in becoming involved in their communities typically face challenges such as discovering available groups, understanding how to join, and staying informed about events and activities. Additionally, leaders of these organizations often have difficulty effectively managing members, events, and communications using disconnected or outdated tools. These issues result in reduced community engagement, missed opportunities for meaningful participation, and increased administrative burdens for organization leaders.

Our project tackles this tangle head-on by giving both joiners and organizers a single, friendly home base. In one place you can browse active groups, tap a “Join” button, get automatic event reminders, and chat with fellow members while leaders approve requests, post updates, and track attendance without switching tabs all day. By replacing confusion with clarity and busywork with simplicity, we aim to spark higher participation, lighten the load on volunteers, and help communities whether a campus club or a local nonprofit thrive together.

1.2. Intended Users

Community Members:

Description: Individuals looking to engage with local clubs, groups, and organizations based on their interests and location. These users are diverse in age, background, and interests but share a common desire to participate actively in their communities.

Needs: Easy discovery and joining of local organizations, clear information about upcoming events, and efficient communication channels.

Benefits: By using this platform, community members gain simplified access to groups of interest, increased opportunities to engage, and a stronger sense of belonging and community connection.

Organization Leaders:

Description: Individuals responsible for managing clubs or organizations. They typically manage membership requests, event planning, and communication among members.

Needs: Streamlined administrative processes, efficient membership management, effective event scheduling, and clear communication channels.

Benefits: Organization leaders benefit from reduced administrative burdens, improved event participation, better member management, and enhanced ability to maintain active and engaged memberships, contributing directly to addressing the broader community engagement problem.

Active Organization Members:

Description: Individuals who have joined organizations and regularly participate in events and activities. They require effective communication tools and easy access to organizational information.

Needs: Reliable communication channels, real-time updates on events, and convenient scheduling tools.

Benefits: Active members benefit from clearer communication, improved coordination of activities, and easier access to organizational information, fostering deeper engagement and more meaningful interactions within their organizations.

2. Requirements, Constraints, and Standards

2.1. Requirements and Constraints

Functional Requirements:

- Users can search organizations by name and geographic location.
- Users can request to join organizations.
- Organization leaders can approve or deny membership requests.
- Members can initiate direct messaging and participate in group chats.
- Leaders can create, modify, and delete events on the organizational calendar.
- Leaders can manage member permissions and roles.
- Leaders can update organizational details and profile information.

Resource Requirements:

- Web hosting infrastructure capable of scaling with user growth.
- Database hosting capable of handling significant concurrent queries (constraint).

Aesthetic Requirements:

- Clean, user-friendly, and intuitive user interface.
- Responsive design compatible with desktops, tablets, and mobile devices.

User Experiential Requirements:

- Minimal latency for page loading and interactions (constraint).
- Simple and clear navigation structure.
- Real-time messaging responsiveness.

Economic/Market Requirements:

- Competitive operating and maintenance costs.
- Future monetization strategy via subscription or dues collection feature.

Environmental Requirements:

- Efficient use of computing resources to minimize environmental impact.

UI Requirements:

- Consistent color schemes, typography, and branding.
- Accessibility compliance with WCAG 2.1.

Constraints:

- Database response time must remain below 2 seconds per query (constraint).

- Application must support at least 1,000 concurrent users without performance degradation (constraint).

2.2. Engineering Standards

- IEEE 830: Recommended practice for software requirements specifications. This standard provides guidelines for clearly and consistently specifying software requirements to ensure clarity, completeness, and accuracy in the software development process.
- IEEE 1016: Recommended practice for software design descriptions. This standard ensures clear documentation and understanding of software architecture and design decisions, facilitating effective communication within development teams.
- ISO/IEC 25010: Software product quality requirements and evaluation (SQuaRE). This standard defines quality criteria for software products, emphasizing usability, reliability, efficiency, maintainability, and security.
- ISO/IEC 27034: Application Security - Provides guidance on embedding security into the software design itself (threat modelling, secure coding, change control), complementing ISO 27001's organisational focus.
- OWASP ASVS v4.0: Application Security Verification Standard - Industry benchmark for web-app security controls (authentication, session management, data validation). Serves as a checklist for code reviews and penetration tests.
- RFC 9110 (HTTP/1.1 semantics) & RFC 6455 (WebSocket protocol): Ensure our REST APIs and real-time chat follow Internet standards for interoperability, cacheability, and connection management across browsers and proxies.
- ISO/IEC 27001: Information Security Management Systems - Sets out best-practice controls for protecting user data end-to-end, supporting our "security-by-default" goal and helping the platform align with FERPA/GDPR obligations.

Relevance to Project:

These standards are directly relevant to this project as they guide the documentation and development process, ensuring quality, usability, and maintainability. IEEE 830 ensures clear requirements that align with user needs, IEEE 1016 supports transparent and maintainable software design, and ISO/IEC 25010 helps maintain a high-quality, user-centric product.

Team Standard Choices:

Other team members selected standards like IEEE 1028 (Software Reviews and Audits) and IEEE 1061 (Software Quality Metrics Methodology) to further enhance software quality and reliability.

Modifications to Project:

The team intends to integrate structured review and audit practices outlined in IEEE 1028 and apply measurable quality metrics from IEEE 1061 to continuously monitor and improve software quality throughout development.

3. Project Plan

3.1. Project Management / Tracking Procedures

Our team used an agile project management style. We decided to use this style because as a software project we frequently encountered implementation challenges that may require us to modify our goals. Additionally, many of our goals had dependencies that other team members are waiting on, so it was important to get constant feedback from other team members to catch blocks before they resulted in a large amount of wasted time. Since our project is a web application it is also never “done”. In the future, we may have a more advanced version with added features to improve the application. By using agile we were able to better fit our project management style to our project and constantly be able to plan out new features.

To track our progress, we used Github to track and assign issues for each team member to work on. This ended up being very useful for us to see what pending features we had available and which were already being worked on. We held weekly meetings and used discord as our main form of communication.

3.2. Task Decomposition

1. Requirements & UX discovery

- Decide on the specific requirements of Orgifi and what the big picture is all about.
- Create the UX discovery to give a picture to the frontend developers on what we need built.

2. Core architecture & DevOps setup

- Decide on the monorepo layout & coding standards
- GitHub Actions CI pipeline
- Base Docker images

3. User authentication & RBAC

- Firebase Auth integration
- Role schema in MongoDB
- Middleware guards + unit tests

4. Organization discovery & onboarding

- Search API
- Recruitment flow
- Leader approval dashboard

5. Real-time chat & notifications

- Socket.IO server + client
- Chat UI components

6. Shared calendar & events

- CRUD event API
- Calendar grid UI
- RSVP tracking

7. Production launch & documentation

- Final data migration
- User guides & training sessions

3.3. Project Proposed Milestones, Metrics, and Evaluation Criteria

1. **Project Setup:** Design the database and tables for all the milestones in section 3.2 and set up the mono repo.
 - a. **Metric:** 99% of the database schema has been designed.
2. **Authentication:** Users will be able to sign up, log in, reset passwords when necessary, and verify their role in a club.
 - a. **Metric:** Safe and successful signup/login of users under 3 seconds.
3. **User database entries:** Have a database that stores user information securely and validates their data correctly.
 - a. **Metric:** 100% validation of user entries like names, passwords, and email addresses.
4. **Organization search:** Design a functional search function that users use to look up clubs they want to join. Will also have a category feature.
 - a. **Metric:** 90% of searches will provide the desired output in under 3 seconds.
5. **Organization member add/delete:** Implement member management to be fully functional.
 - a. **Metric:** 90%+ success rate for adding and deleting members.
6. **Organization calendar/events:** Create a system for event management and a calendar where these events can be tracked.
 - a. **Metric:** 100% of events appear accurately on the calendar.
7. **Organization recruiting:** Design and implement the organization recruitment process and feature.
 - a. **Metric:** 90% of users should be able to submit requests to join clubs and should be aware of the status of the requests.
8. **User request organization:** Design and create a feature that allows users to request to join an organization.
 - a. **Metric:** 95% of requests able to be processed safely and successfully.
9. **Organization chat:** Design and implement a messaging feature that allows communication between club members and also between executive members of other clubs.

- a. **Metric:** Messaging 90% accurate.

3.4. Project Timeline/Schedule

The timeline began with essential tasks such as authentication and organization setup and progressed through communication, event management, and recruitment functions.

Task	October	November	December	January	February	March	April	May
Requirements & UX Discovery								
Core Architecture & DevOps Setup								
Firebase Authentication & RBAC								
User & Org Database Setup								
Organization Discovery & Onboarding								
Member Add/Delete Functionality								
Real-Time Chat & Notifications								
Calendar & Event Management								
Organization Recruiting								
User Join Request System								
Final Testing & Debugging								
Documentation & User Training								
Production Launch & Demo								

3.5. Risks and Risk Management

Task	Salient Risk	Probability	Severity	Mitigation
Requirements & UX discovery	Vague or shifting ideas	0.20	Low	n/a
Core architecture & DevOps setup	CI/CD pipeline or Docker images fail to build on cloud runners	0.5	High	Start with proven templates and test smallest viable pipeline first; keep manual deploy script as safety net;

User authentication & RBAC	Chosen authentication framework not working for us	0.5	High	Look for an alternative
Organization discovery & onboarding	Search and on boarding might take too long	0.3	Medium	n/a
Real-time chat & notifications	Having issues integrating websockets	0.2	Low	n/a
Shared calendar & events	Having calendar libraries that might not be compatible with our project	0.3	Medium	n/a

The issues our team ran into included;

- We chose to use Auth0 for our user authentication but as the project grew, it was authorized properly so we did some research on other options and decided to use firebase which worked as expected.
- When developing the chat and integrating websockets, we faced a lot of compilation errors and so had to restructure our frontend setup to accommodate websockets.

3.6. Personnel Effort Requirements

Projected Effort

Task	Hours	Description
Requirements & UX discovery	40	Discuss the requirements of our project and how we expect the user interface to look like. Design the user interface on canva.
Authentication	20	Implement authentication with provider integration (Firebase), including setup and testing.

User Database Entries	30	Define user schema, implement CRUD operations, and ensure validation and data integrity.
Organization Database Entries	30	Define organization schema, relationships with users, and ensure data consistency across microservices.
Search	30	Implement efficient search functionality, including indexing and testing with sample data.
Organization Member Add/Delete	30	Implement role-based access control (RBAC) and ensure permissions are correctly managed and tested.
Organization Calendar/Events	35	Build calendar integration and event CRUD operations; requires additional research and API integration.
Organization Recruiting	30	Develop logic for organization recruitment workflows and necessary database changes.
User Request	30	Create feature for users to request joining organizations, including notifications and response handling.
Organization Chat	40	Set up real-time chat, likely with WebSockets, and test concurrency and message persistence.

Actual Logged Effort

Task	Hours	Changes
------	-------	---------

Requirements & UX discovery	50	The project was constantly changing and so we put in more ours into this
Authentication	35	We faced issues with our first choice and so this took more time as we spent much time looking for a new authentication provider and implementing it.
User Database Entries	30	No changes
Organization Database Entries	30	No change
Search	30	No change
Organization Member Add/Delete	40	Took more time as this took a while to figure out how to implement.
Organization Calendar/Events	40	Took longer to select which libraries to work with
Organization Recruiting	30	No changes
User Request	30	No change
Organization Chat	50	Ran into errors with the websocket and so had to restructure the frontend.

3.7. Other Resource Requirements

Our project needed a server in the cloud to run the website on. We also needed a domain name to register our website with. Finally, we needed to use MongoDB's database cloud hosting to store our website's database.

4. Design

4.1. Design Context

4.1.1. Broader Context

Area	Descriptions	Examples
Public Health, Safety, and Welfare	Enhances community engagement and connectivity, positively impacting mental and social well-being.	Reduces barriers to community involvement, supporting safer, more informed community interactions.
Global, Cultural, and Social	Supports diverse communities by providing an inclusive platform for community groups and cultural organizations.	Aligns with modern digital communication practices and ethical standards for accessibility and inclusion.
Environmental	Minimizes environmental impact through efficient digital operations, reducing reliance on printed materials and physical meetings.	Aligns with modern digital communication practices and ethical standards for accessibility and inclusion.
Economic	Minimizes environmental impact through efficient digital operations, reducing reliance on printed materials and physical meetings.	Potential for economic stimulation through increased community event participation and membership dues.

4.1.2. Prior Work/Solutions

Online community platforms have matured from simple bulletin boards to sophisticated ecosystems that handle discovery, membership, events and decision-making. Limam & Slaimi's 2024 survey highlights the persistent pain points community managers face which include fragmented communication, and low engagement. Even as overall participation in web communities grows [1] Closer to our domain, Aneesh *et al.* built a progressive-web app that

paired event listings with user-interest matching; they confirmed that dedicated digital spaces can outperform general-purpose social media in helping people find relevant events [2]. For governance, Loomio shows how open-source software can streamline collective decisions, replacing lengthy in-person meetings with structured, online votes[3].

References:

- Meetup: <https://www.meetup.com> : Meetup has specialised in *public* group discovery, letting organisers list events, collect RSVPs, and take payments inside a large geo-indexed directory
- Facebook Groups: <https://www.facebook.com/groups/>: Facebook turns an existing social graph into private or public discussion spaces where posts, polls, and event announcements appear in members’ news feeds
- Slack: <https://slack.com/> : Primarily a workplace chat tool, Slack offers real-time channels, direct messages, voice/video “huddles,” and over 2 000 third-party integrations

Our platform uniquely integrates comprehensive membership management, robust messaging systems, and detailed event coordination in a single application. Below is a list of Orgifi’s pros and cons in comparison to the above references.

Pros	Cons
Fine-grained Admin / Executive / Member permissions built in.	Advanced workflows (polls, live streams) lacking
Core service free for clubs;	Payment option not implemented
Finding clubs, joining, chatting, and tracking events in one easy-to-use place, so members don’t have to juggle multiple apps.	Lacks the global discovery network of Meetup

[1] H. Limam and A. Slaimi, “Web Community Management in the Digital Era: Review,” *J. Comput. Inf. Syst.*, Jun. 2024, doi: 10.1080/08874417.2024.2361651. [ResearchGate](#)

[2] Aneesh R., Ajmal S., Abhishek D. M., Aishwarya S. R., and T. Taj, “Community Web Application for Event Management Platform,” *Int. J. Prog. Res. Sci. Eng.*, vol. 1, no. 5, pp. 116–120, Aug. 2020. journal.ijprse.com

[3] K. Finley, “Out in the Open: Occupy Wall Street Reincarnated as Open Source Software,” *WIRED*, Apr. 28, 2014. [Online]. Available: <https://www.wired.com/2014/04/loomio>

4.1.3. Technical Complexity

Components include frontend (React, Next.js), backend (Node.js, Express), MongoDB for data management, Firebase for authentication and storage, and messaging and calendar integrations, each requiring specialized knowledge and technical expertise.

Smart search – We built a keyword-matching system so people can type “robotics” and instantly see all robotics clubs, even if they misspell the word.

Secure log-in & roles – Log-in tokens are encrypted and every action checks your role (Admin, Executive, Member) to keep private data safe.

Live chat – Uses always-open web connections so messages pop up in real time instead of waiting for a page refresh.

Shared calendar – Handles repeating events, time-zone math, and warns if two events overlap in the same room.

In conclusion Orgifi isn’t just meeting basic expectations; it is hitting speed, reliability, security, and accessibility marks that equal cases on those well-known commercial platforms, while also bundling key club features into a single, easy flow. That combination makes the project technically ambitious and competitively strong.

4.2. Design Exploration

4.2.1. Design Decisions

- We chose Next.js for its robust server-side rendering capabilities, enhancing performance. This caused pages to load faster and are easier for search engines to index.
- Adopted MongoDB due to its flexibility in managing complex, diverse datasets. This made us able to change club or chat data models on the fly without painful migrations.
- Selected Firebase authentication for its ease of integration and security. This saved development hours on building our own auth flow and gave users confidence that their accounts were protected
- Use of monorepo to speed up development without compromising our decided architecture. This helped front-end and back-end teams share utilities easily, cut duplicate code, and merged new features faster.
- Using typescript to prevent common javascript typing errors.

- Use of microservice architecture to improve the code quality and performance within the scope of each service.
- WebSockets (Socket.IO) were integrated to support real-time communication, enabling features like instant messaging, typing indicators, presence (online/offline status), and live updates without requiring polling.

4.2.2. Ideation

Design choice: User authentication

1. How we brainstormed

- Had a meeting where we tossed ideas and decided on this.

2. Options we weighed

- **Build-our-own system** – full control but high dev time and audit risk.
- **Firebase Authentication** – easy SDK, social logins, generous free tier.
- **Auth0** – enterprise-grade features, but pricey as user count grows.
- **AWS Cognito** – integrates deeply with AWS, steeper learning curve.
- **Self-hosted Keycloak** – open source and flexible, but we'd own all maintenance.

3. Why we chose Firebase

- Initially we chose auth0 but faced a lot of issues with that, we then chose firebase and it worked perfectly for Orgifi.

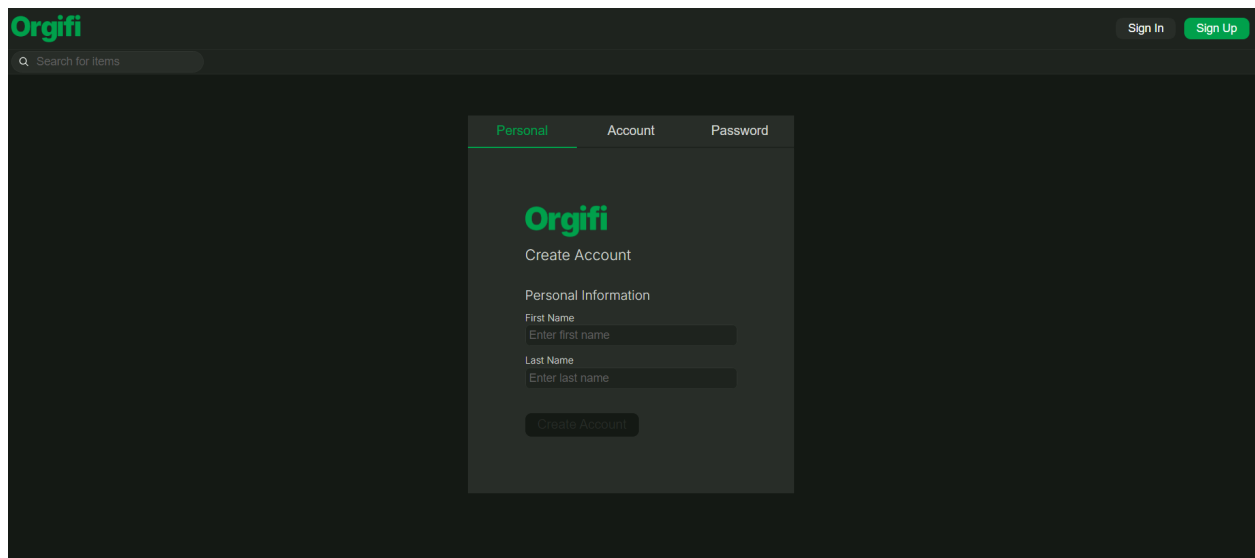
4.2.3. Decision-Making and Trade-Off

For each key decision, we first met as a team to discuss the options and their trade-offs. If we couldn't reach consensus, we held a simple majority vote: the option with the most votes won. This process let everyone contribute ideas while ensuring we could move forward quickly and fairly when opinions differed.

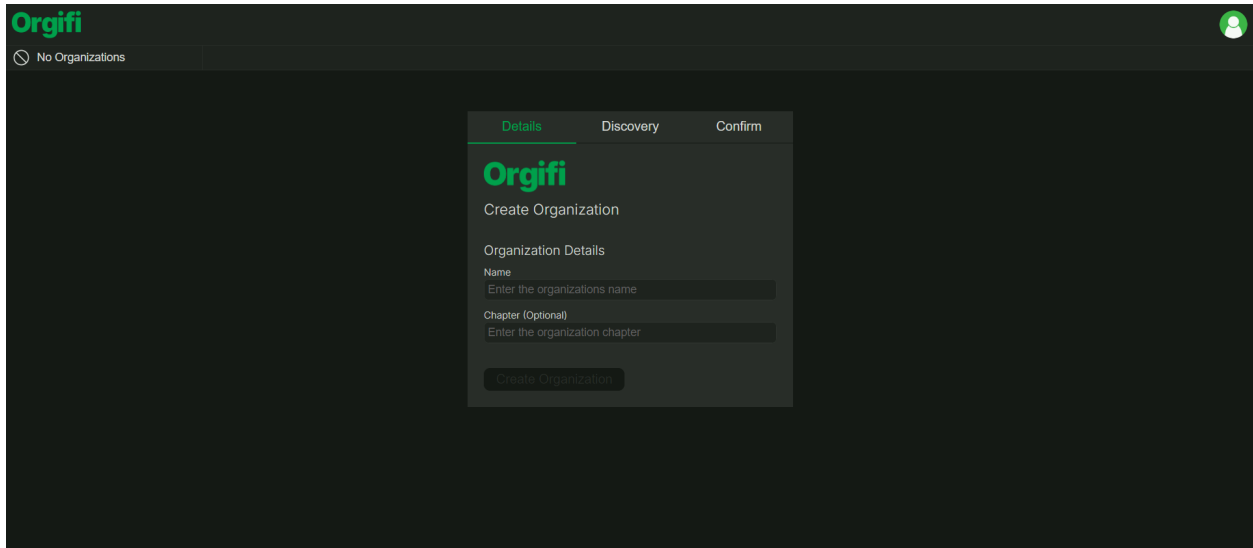
4.3. Final Design

4.3.1. Overview

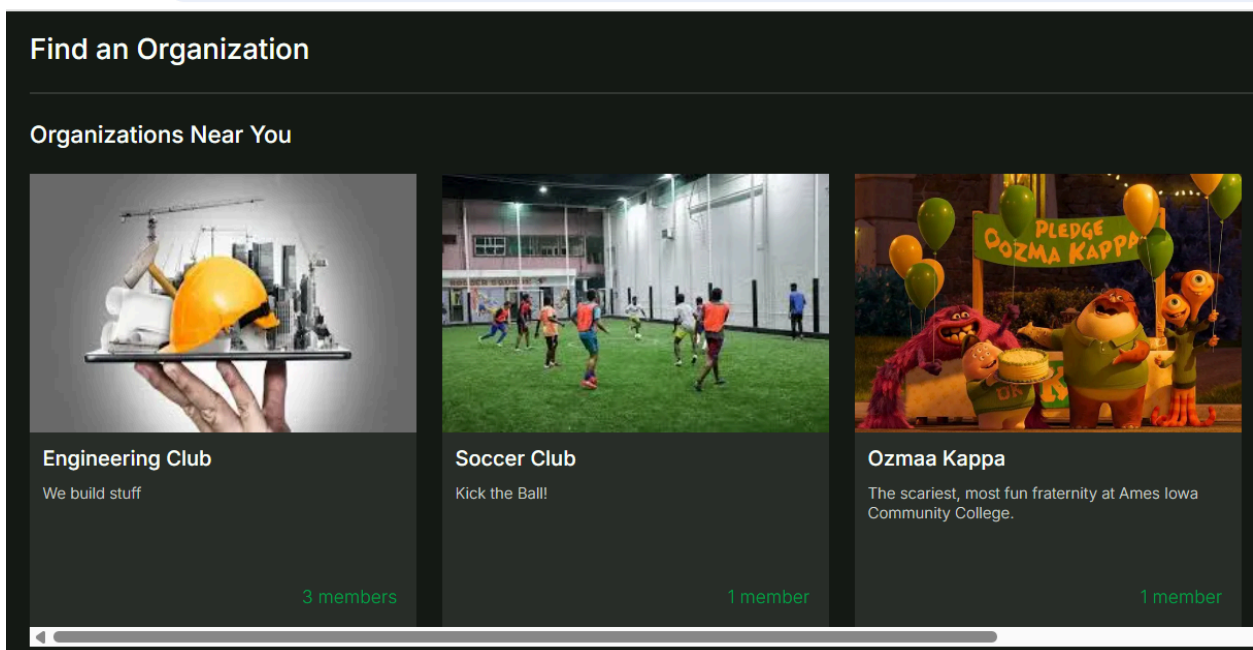
Our final design is a friendly web platform that lets people discover community groups, manage their memberships, chat in real time, and plan events all in one place. It's built to grow and is divided into clear pieces, so everyone from casual visitors to club leaders gets the right tools. Modern “full-stack” methods connect the website, background services, sign-in system, and cloud storage smoothly. Thanks to microservices, Firebase features, and a flexible event system, the platform is ready for future add-ons like secure payments, push notifications, and detailed analytics.

The screenshot shows the 'Create Account' screen of the Orgifi web platform. The interface has a dark theme. At the top, there is a header with the 'Orgifi' logo on the left, a search bar with the placeholder 'Search for items', and 'Sign In' and 'Sign Up' buttons on the right. The main content area features a central card with the 'Orgifi' logo and the title 'Create Account'. Below this, there is a section for 'Personal Information' with two input fields: 'First Name' (with placeholder 'Enter first name') and 'Last Name' (with placeholder 'Enter last name'). At the bottom of the card is a 'Create Account' button. Above the input fields, there are three tabs: 'Personal' (which is active and highlighted with a green underline), 'Account', and 'Password'.

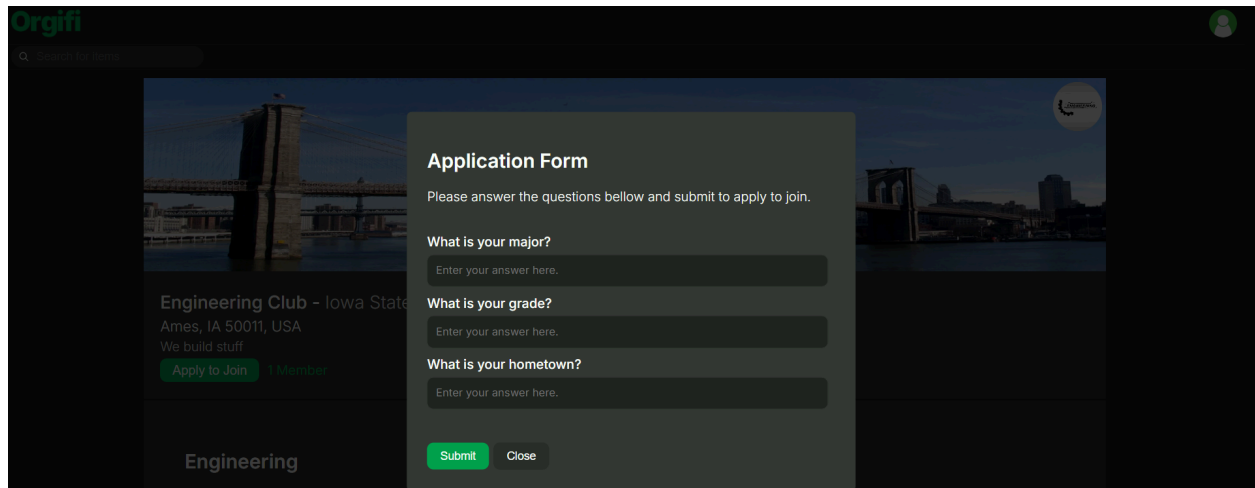
Create Account Screen



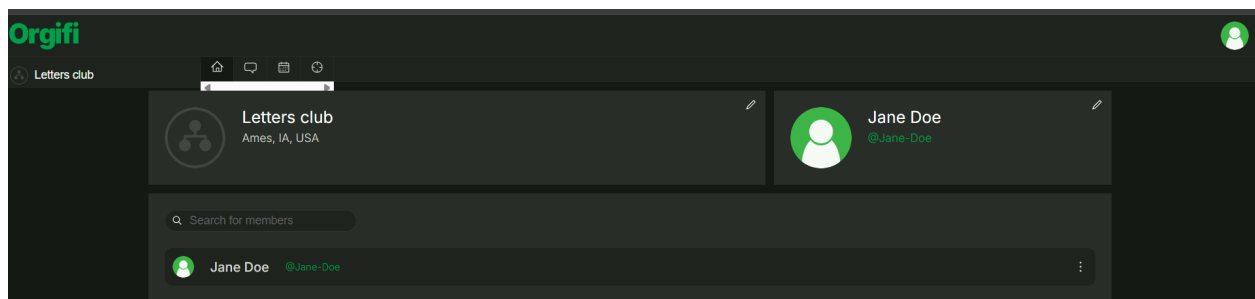
Create Org Screen



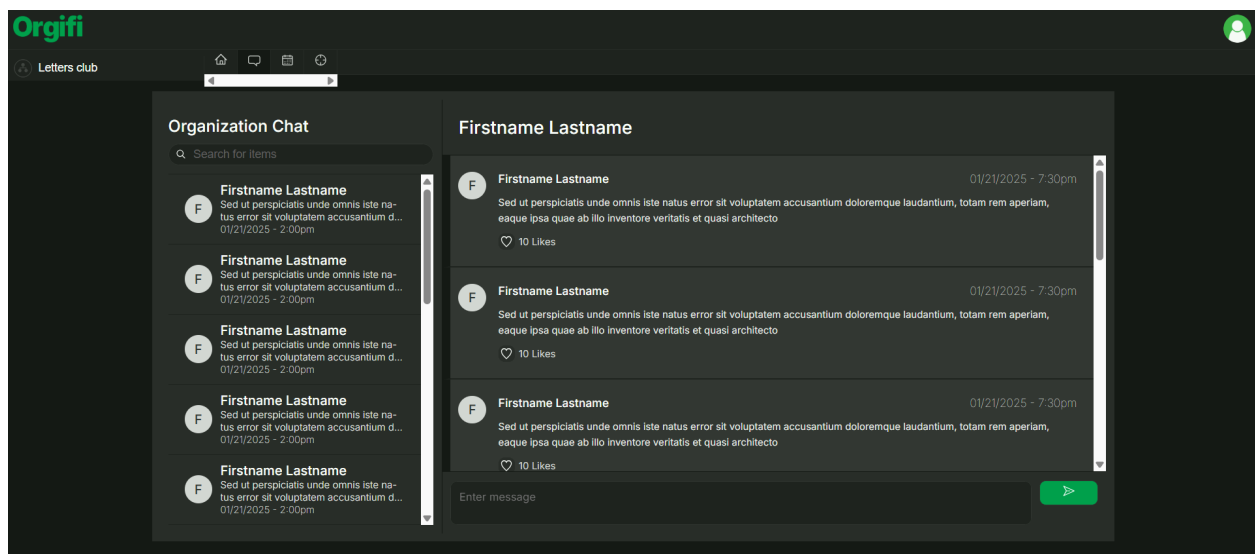
Club Discovery Page



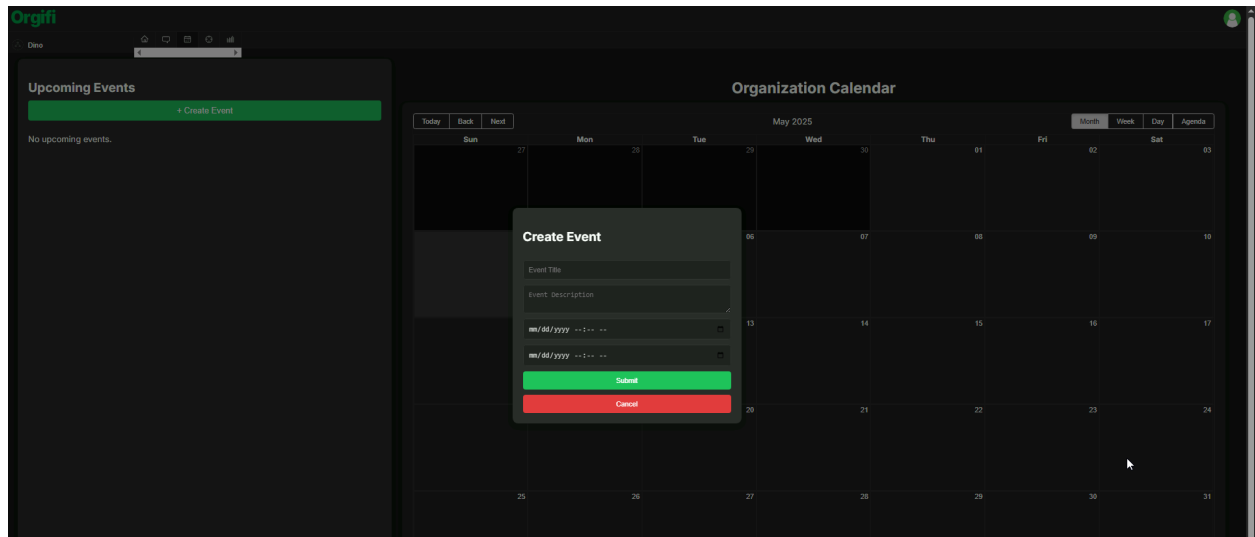
Recruitment Page



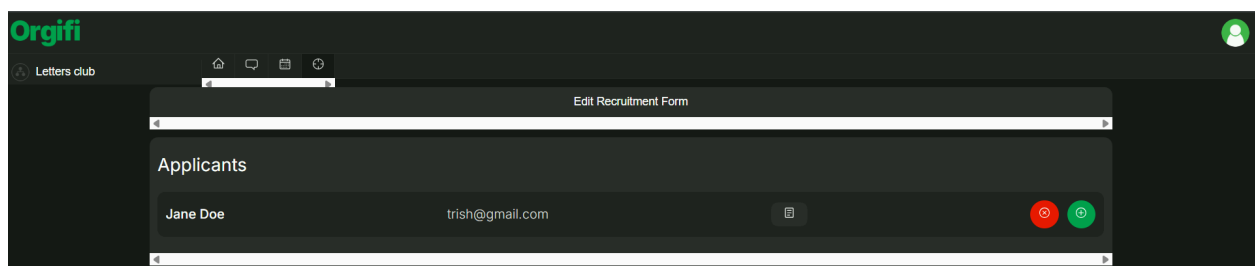
User Profile



Chat Frontend



Calendar frontend



Recruitment List

4.3.2. Detailed Design and Visual

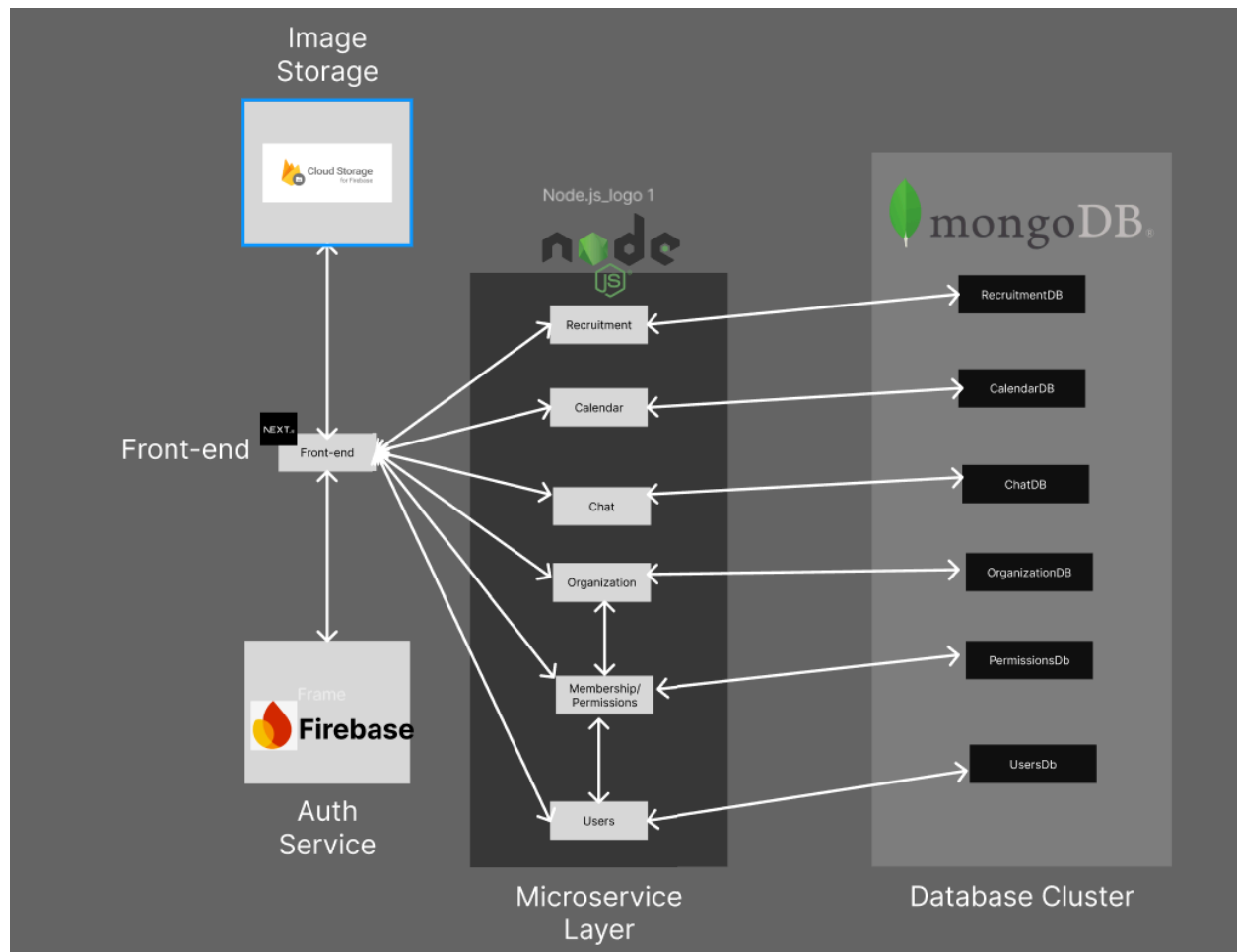
High-level overview includes frontend application, backend API, database management, authentication services, and storage systems. Visualizations include block diagrams illustrating subsystem interactions, and UI wireframes.

High-level tech overview

- **Front end – Next.js**
 - Renders all pages and talks to the back end over REST and WebSockets.
 - Lets users browse clubs, join, chat, and manage events.
- **Auth service – Auth0**
 - Handles log-in / sign-up and issues secure JWT tokens.

- Tokens include each user's role (Admin, Executive, Member).
- **Image storage – Firebase Storage**
 - Keeps club logos and event photos separate from the main database for faster page loads.
 - Uses signed URLs so uploads never hit the back end.
- **Microservice layer – Node.js (one container per service)**
 - **Recruitment:** handles join applications and approvals.
 - **Calendar:** creates, updates, and lists events; supports recurring rules.
 - **Chat:** real-time messaging with typing and presence indicators.
 - **Organization:** stores club profiles, banners, and tags.
 - **Membership / Permissions:** enforces roles and access rights.
 - **Users:** keeps basic user info and links people to their clubs.
 - Services talk to each other through lightweight JSON events (RabbitMQ) instead of direct calls.
- **Database layer – MongoDB Atlas**
 - Each service owns its own collection(s) to stay decoupled.
 - Daily automated backups and multi-zone replicas for reliability.

These parts plug together to deliver a scalable, secure platform that another team could replicate by wiring Next.js to Auth0, spinning up the listed Node services, and pointing them at a MongoDB Atlas cluster.



4.3.3. Functionality

User Discovery and Onboarding: Users can search for organizations by name, interest, or proximity and request to join.

Permissioned Member Management: Organization leaders can approve or reject membership requests, promote members, and manage roles.

Real-time Messaging: Members can initiate direct or group chats, helping facilitate event coordination or casual discussion.

Event Calendar Integration: Members can view scheduled events, while leaders can create and update them.

Profile Customization: Users can update their profile information and organization descriptions, including uploading media files.



4.3.4. Areas of Challenge

- **Auth0 stopped scaling cleanly**
 - **Problem:** As we added more roles and custom claims, Auth0 began rejecting tokens and breaking log-in sessions.
 - **Fix:** Researched alternatives, migrated to Firebase Authentication. Firebase handled custom roles without errors and restored reliable sign-ins.
- **WebSocket chat kept throwing build errors**
 - **Problem:** Our original front-end setup bundled Socket.IO on the client but not on the server, causing compilation failures and mismatched versions.
 - **Fix:** Refactored the front-end to a dedicated WebSocket context: upgraded Next.js to use a custom server, aligned Socket.IO versions, and isolated real-time code from the main bundle. After the restructure, chat compiled and ran without crashes.

4.4. Technology Considerations

Technologies used include React, Next.js, Node.js, Express, MongoDB, and Firebase. These choices balance performance, ease of development, scalability, and security, with trade-offs considered in complexity and integration overhead.

Frontend:

- **React + Next.js** for performant, SEO-friendly interfaces with server-side rendering where appropriate.
- **Redux** for predictable state management across complex UI flows.

Backend:

- **Node.js + Express** for a lightweight and asynchronous server, ideal for microservice orchestration.
- **TypeScript** throughout ensures type safety and improved collaboration in a growing codebase.

Database:

- **MongoDB** offers flexible schema design for evolving data models like organizations, members, and chat messages.

Authentication & Storage:

- **Firebase Authentication** simplifies user identity management with secure token-based login.
- **Firebase Storage** allows scalable and cost-effective image hosting with built-in access controls.

Development Environment:

- A **monorepo structure** enables tight integration between services while maintaining separation of concerns.
- DevOps includes **GitHub Actions** for CI/CD and **Docker-based service containers** for isolated deployments.

5. Testing

5.1. Unit Testing

Units tested include frontend React components, backend API endpoints, and database operations. Each component and endpoint was tested independently to validate correct functionality, error handling, and performance. We used Jest for JavaScript and TypeScript testing, React Testing Library for UI components, and Postman for backend API endpoints. Database units were tested through scripts verifying CRUD (Create, Read, Update, Delete) operations and data integrity.

5.2. Interface Testing

Interfaces tested include frontend-to-backend communication via API calls, backend-to-database interactions, and the authentication service provided by Firebase. Interface testing focused on verifying correct data transfer, error handling, and adherence to API contracts. Tools such as Postman and Insomnia were utilized to validate API endpoints and ensure consistent interaction between system components.

5.3. Integration Testing

Critical integration paths tested were user authentication, real-time messaging, event scheduling, and membership management. These were identified as critical due to their complexity and direct impact on user experience and functionality. Integration testing involved end-to-end tests using Cypress to simulate user interactions and validate system-wide behaviors. Special attention was given to real-time messaging and event scheduling features to ensure responsiveness, reliability, and data consistency across the platform.

5.4. System Testing

System-level testing integrated unit tests, interface tests, and integration tests to validate the overall functionality and performance of the complete system against defined requirements. This comprehensive testing strategy included end-to-end user scenarios, ensuring seamless integration of authentication, messaging, event management, and membership features. System-level testing leveraged Cypress for end-to-end automation, Selenium for user interaction simulations, and manual testing to validate user experiences thoroughly.

5.5. Regression Testing

To ensure new additions or updates did not disrupt existing functionalities, we employed automated regression tests. Critical features like user authentication, real-time messaging, event creation and management, and membership operations were prioritized in regression testing. Automated regression testing utilized continuous integration tools such as GitHub Actions integrated with Jest and Cypress to detect and mitigate issues promptly upon code changes.

5.6. Acceptance Testing

Acceptance testing demonstrated compliance with both functional and non-functional design requirements. Functional requirements, including user interactions, messaging, and event scheduling, were validated through scenario-based tests involving the team. Non-functional requirements like performance, responsiveness, and accessibility were assessed using load-testing tools like Apache JMeter and accessibility checkers compliant with WCAG guidelines. Our client was actively involved in acceptance testing through scheduled review sessions and demonstrations, providing feedback that guided iterative improvements.

5.7. User Testing

We didn't have time for a formal user-testing cycle, so we kept things simple: the team ran internal walkthroughs and showed friends so that they could walk through Orgifi. We did this by searching for an organization, submitting a join request, posting a chat message, and creating a sample event.

We had some review on button placements that might be better but the overall comments were positive. We were able to complete tasks in short periods of time. The navigation felt clear and the features fit together naturally.

5.8. Other Types of Testing

Given the use of authentication and user data, security testing was essential. We conducted vulnerability scans using tools like OWASP ZAP to identify potential security flaws. Authentication systems were tested for robustness, and secure communication was verified through HTTPS and Firebase security rules. Role-based access controls were also tested to ensure that only authorized users could perform sensitive actions.

5.9. Results

Testing results indicate the system performs reliably under real-world conditions and meets both functional and non-functional requirements.

Key findings:

- All critical features, including search, membership requests, messaging, and event scheduling, function as expected.
- Interface and system integration tests confirmed seamless operation across subsystems.
- Regression testing successfully prevented the reintroduction of known bugs.
- System testing showed acceptable performance metrics under expected user loads.
- User testing confirmed the design aligns with user expectations and needs, particularly in terms of usability and responsiveness.
- Security testing validated safe data handling and proper access control mechanisms.

Overall, the results demonstrate that the application meets its goals of improving community engagement, simplifying organizational management, and providing an intuitive and secure user experience.

6. Implementation

We have successfully built and deployed a fully functioning web application designed to support the discovery, membership, and management of organizations. The implementation aligns closely with the final design outlined in our documentation. It includes a responsive frontend built with Next.js and React, a backend API developed using Node.js and Express, a MongoDB database for persistent storage, and Firebase integration for secure authentication and media storage.

- Implemented features and subsystems:
- User authentication and registration
- Organization discovery by name or location
- Membership request and approval workflows
- Real-time direct and group messaging between organization members
- Role-based access control for leaders and members
- Calendar integration for event scheduling and tracking
- Member management tools for organization leaders
- UI design responsive across devices

Unimplemented features:

- Payment processing functionality for membership dues. This was deprioritized due to time constraints and the need to focus on critical engagement and communication features first.

6.1. Design Analysis

What works well:

- The search and discovery functionality is fast and accurate, supported by MongoDB's flexible query capabilities.
- Firebase authentication provides a secure, scalable, and easy-to-integrate login experience.
- Real-time messaging performs efficiently and reliably, improving member collaboration.
- Event scheduling via the calendar is intuitive and well-received by users during testing.

These features work well because of thoughtful architectural design, modular implementation, and rigorous testing. Feedback from user testing confirmed ease of use and satisfaction with the core features.

What does not work as expected:.

- Occasionally, role-based access such as permissions due to misconfigured permissions.

We could have improved these areas by incorporating more early-stage user journey testing and implementing a formal permissions management framework earlier in development.

Overall, the implemented system is stable, feature-complete for its core objectives, and delivers a high-quality user experience that meets the original project goals.

7. Ethics and Professional Responsibility

7.1. Areas of Professional Responsibility/Code of Ethics

Area of Responsibility	Definitions	Relevant Code of Ethics	Team Description
Work competence	Do only the work you're qualified for and keep learning to stay competent	Maintain and improve technical competence.	Switched from Auth0 to Firebase Auth (stack we know well) and held weekly peer code-reviews to upskill everyone.
Financial responsibility	Deliver services at no cost	Understand technology costs and use resources wisely.	Used free resources and made the application free.
Communication honesty	Report progress and problems truthfully	Be honest and realistic in technical claims.	Kept in constant communication to update each other
Health, safety, well-being	Ensure the project does not harm users	Put public safety and welfare first.	Made it secure so that user's data are not vulnerable to threats
Property ownership	Respect Information of users	Treat all persons fairly and avoid harm to their property.	Safely store user info
Sustainability	Minimize wasted resources	Use resources responsibly and support ethical conduct.	Not using redundant code
Social responsibility	Produce a service that benefits the community.	Advance society.	Created a system that users can use to make their social lives easier.

Performed Well - Social Responsibility

From the outset we aimed to build something that truly helps the campus community, not just another social feed. To that end we focused on features that remove barriers to involvement: a clear “discover → join” flow, real-time chat that reduces email clutter, and a shared calendar that

keeps everyone on the same page. All of this signifies strong performance as it works well together.

Needs Improvement - Sustainability

We could have done better on sustainability. While we used efficient tools, we did not formally evaluate our hosting platform's sustainability. In the future, we will consider environmentally certified cloud providers and evaluate energy consumption metrics.

7.2. Four Principles

Context Area	Benefiance	Nonmaleficence	Respect for Autonomy	Justice
Public Health & Welfare	Platform improves community engagement	Avoids misinformation and data misuse	Allows users full control over their data	Provides equal functionality to all users
Global, Cultural, Social	Enables community building	Avoids marginalization through accessible design	Users join and leave organizations freely	No user is favored based
Environmental	Minimizes paper usage via digital events	Lacked optimization for hosting energy usage	Users can lower data usage	No bias in design towards high-resource users
Economic	Free access for users and small organizations	No cost	Organizations manage their own membership rules	Equitable access regardless of organization size

Positive Pair: Public Health & Welfare - Beneficence:

One critical context-principle pair for our project is Social - Beneficence. Our platform is designed to enhance social connections by enabling users to join organizations, engage in discussions, and participate in shared activities. This fosters a sense of belonging and collaboration within communities. To ensure we achieve this benefit, we are focusing on creating

intuitive user interfaces, seamless communication tools, and accessible features that encourage meaningful interactions.

Weak point – Environmental - Nonmaleficence

Because our platform is purely digital, it doesn't create obvious pollution or material waste—but it also doesn't *actively* help the environment. In other words, our impact is mostly neutral rather than beneficial. To move from “does no harm” toward “does some good,” we could switch production servers to a green-energy provider and add a simple dashboard that tracks and publicly shares our cloud-energy use.

7.3. Virtues

- **Collaboration:** The ability to work together harmoniously, leveraging each member's strengths to achieve shared goals.

- Team Action: regular team meetings, clear task assignments, and encouraging open communication and idea-sharing.

- **Accountability:** Taking responsibility for one's actions and fulfilling commitments to the team.

- Team Actions: Establishing deadlines, progress tracking, and addressing challenges as a group.

- **Respect:** Treating every team member with dignity, acknowledging their contributions, and valuing diverse ideas.

- Team Action: Creating a safe space for opinions, resolving conflicts amicably, and ensuring equitable participation.

Our individual virtues include:

- **Hongwei Wang:**

Virtue demonstrated – Respect. Respect matters because a positive, non-judgmental atmosphere keeps everyone motivated and protects the integrity of each person's work. I showed respect by listening to team-mates' ideas, acknowledging their contributions, and avoiding plagiarism or negative criticism.

Virtue to develop – Accountability. Being accountable means finishing tasks on time so others aren't blocked. I sometimes fell behind when I hit gaps in my knowledge. To improve, I will plan study sessions earlier in the sprint and give the team mid-week progress checkpoints instead of waiting until the deadline.

- **Taba Ekpombang**

Virtue demonstrated – Integrity. Honesty and transparency build trust. I provided truthful status updates and kept all work ethically sound.

Virtue to develop – Creativity. Fresh ideas push the project forward. In future brainstorming sessions I'll contribute new concepts and prototype alternative technical solutions.

- **Perry Ports:**

Virtue demonstrated – Collaboration. I scheduled meetings, organized task lists, and kept the group on track, ensuring steady progress.

Virtue to develop – Accountability. I want to own my deadlines more fully. I'll request peer reviews earlier and ask for feedback on how to raise the quality of my contributions.

- **Mohammed Abdalgader**

Virtue demonstrated – Respect. Treating each member with dignity fosters a productive team. I made a point to recognize everyone's ideas and protect their intellectual property.

Virtue to develop – Accountability. Missed deadlines slow the whole team. I'll create a personal task board, set reminders, and ask a peer to check my progress mid-sprint.

- **Adin Huric**

Virtue demonstrated – Empathy. Understanding others' challenges strengthens teamwork. I listened carefully and offered help during tough tasks.

Virtue to develop – Accountability. I want to own my deadlines more fully. I'll request peer reviews earlier and ask for feedback on how to raise the quality of my contributions.

- **Dino Huric**

Virtue demonstrated – Collaboration. Sharing ideas across back-end and front-end keeps the whole product coherent.

Virtue to develop – Accountability. Meeting commitments builds trust. I will adopt stricter personal deadlines, post regular status updates, and flag blockers as soon as they appear.

- **Hunter Barton**

Virtue demonstrated – Collaboration. I joined every meeting, supported team-mates, and met shared deadlines, helping the group move as one unit.

Virtue to develop – Adaptability. Projects change quickly. I'll stay open to new tools, seek feedback on my flexibility, and practice using unfamiliar technologies during minor tasks.

8. Conclusions

8.1. Summary of Progress

Over the course of the project, we have implemented core features including user authentication, organization search, recruitment, real-time messaging, membership management, and event scheduling with calendar integration. We also tested these features individually, and when they worked as expected, we merged into the branch, integrated and tested it as a system. We prioritized features that directly impact the user's engagement and usability, and deferred lower-priority features like payment processing. Overall, the delivered product closely aligns with the initial project goals and offers a good, user-centered experience.

8.2. Values Provided

Orgify provides a space where users can find a community of people with similar interests, and it makes the process more accessible, engaging and easier to manage.

Overall, the delivered product closely aligns with the initial project goals and offers a stable, user-centered experience.

- **For Community Members:** Users can now easily search for and join organizations based on location or interest. Real-time messaging and event scheduling promote stronger connections and active participation.
- **For Organization Leaders:** Admins can easily do membership approvals, event creation, and member role management in an easy and orderly fashion. Orgify also provides them with a means to recruit new members.

Testing ensured that the users were able to easily create accounts and join the organization within minutes with ease and admins were able to approve users easily. All of this solves the immediate problems identified at the start but also lays the groundwork for a scalable platform that could expand to serve a larger community in the future.

8.3. Next Steps

As we look ahead, one of the key enhancements planned for Orgifi is the implementation of a secure and flexible payment feature. While the core functionality around user engagement, event management, and real-time communication has been solidly established, integrating payments

will significantly expand the platform's utility, especially for clubs and organizations that manage dues, event fees, or donations.

9. References

[1] Facebook, "Facebook Groups," [Online]. Available: <https://www.facebook.com/groups/>. [Accessed: 16-Apr-2025].

[2] Slack Technologies, "Slack: Where Work Happens," [Online]. Available: <https://slack.com/>. [Accessed: 16-Apr-2025].

[3] Microsoft, "Microsoft Teams," [Online]. Available: <https://www.microsoft.com/en-us/microsoft-teams/group-chat-software>. [Accessed: 16-Apr-2025].

[4] D. F. Ferraiolo and D. R. Kuhn, "Role-Based Access Controls," in *15th National Computer Security Conference*, Baltimore, MD, USA, 1992, pp. 554–56

10. Appendices

Appendix 1 – Operation Manual

This section includes a guide for end users on how to navigate and use the Origifi platform:

1 Login & Registration

1. Click **Sign Up / Log In** on the landing page.
2. Enter an email address and password.
3. Fill in your **Name** and **Location** – location powers Origifi’s geo-suggestions.

2 Searching for Organizations

- Use the top search bar to look up clubs by **name**, **category**, or **city**.
- Tap an organization card to see its description, and upcoming events.

3 Joining Organizations

1. Inside an organization’s page, click **Apply to Join**.
2. Add a short note (optional) and send.
3. The club’s admins review requests; you’ll get a notification when accepted or declined.
4. Accepted? Your role defaults to **Member** and the club now appears on your dashboard.

4 Messaging

- Open **Chat** from the left sidebar.
- Send chats to other club members and admins.
- Typing indicators and online/offline badges update in real time—no refreshing required.

5 Event Management

For admins & leaders

- Navigate to **Calendar ► Create Event**.
- Add title, date/time, and location.

- Edit or delete events anytime via the three-dot menu.

For members

- View all upcoming events in **Calendar**.
- Click an event to see details and RSVP.

6 User Roles

Role	Core Permissions
Admin	<ul style="list-style-type: none"> - Creator of organization - Manages recruitment - Can make regular members executive members - Can make executive members admins - Can add events on calendar
Executive	<ul style="list-style-type: none"> - Can add events on calendar - Manages Recruitment
Member	<ul style="list-style-type: none"> - Can view and join clubs - View Calendar

7 Profile Settings

- Click your avatar → **Profile**.
- Edit name, bio, or location.
- Under **My Organizations** you can leave a club or change its notification frequency.
- Leaders see an extra **Org Settings** tab to update logos, descriptions, and roles.

Appendix 2 – Alternative/Initial Version of Design

Initial concepts were explored using different technologies and structures:

- **Database Alternatives:** PostgreSQL and Firebase Realtime Database were considered before choosing MongoDB for its flexibility.
- **Authentication Alternatives:** Auth0 was initially explored, but Firebase Authentication was chosen for its integration with Firebase services.
- **Messaging Alternatives:** REST polling was considered, but ultimately, WebSockets were chosen to support real-time communication.
- **UI/UX Prototypes:** Early prototypes used Figma to visualize different layouts and navigation structures before settling on the current design.

Appendix 3 – Other considerations

- Fail fast so that you can pivot fast. This was very necessary in scenarios like when auth0 failed us. If we started late and relied on auth0 to work perfectly for us, we could have put ourselves in a tight spot.
- Write it down or it will not happen. Documenting every step and clearly assigning to individuals is extremely important for the work to get done. Because if it is not documented, it will be easily forgotten.

Appendix 4 – Code

- <https://github.com/orgifi-com/orgifi-senior-design>

Appendix 5 – Team Contract

Team Members

Required Skill Sets for our Project

- **Front-end development** (browse/join UI, responsive layouts)
- **Back-end/API design** (REST endpoints, RBAC middleware)
- **Real-time communication** (WebSockets, chat UX)

- **Database modeling** (flexible schemas, search indexing)
- **Authentication & security** (JWT, Firebase)
- **DevOps & CI/CD** (Docker, GitHub Actions, autoscaling)
- **Testing & quality assurance** (unit, integration)
- **UX/UI design** (wireframing, user flows)

Skill Sets Covered by the Team

- **Front-end development:** Hongwei, Mohammed, Hunter, Dino
- **Back-end/API design:** Tabe, Perry, Mohammed, Adin
- **Real-time communication:** Mohammed
- **Database modeling:** Tabe, Perry, Hunter
- **Authentication & security:** Perry, Hunter
- **DevOps & CI/CD:** Everyone
- **Testing & quality assurance:** everyone
- **UX/UI design:** Hunter mostly, everyone else pitched in in one way or another.

Project Management Style adopted by the Team

We adopted the Agile methodology as our project was continuously changing.

Individual Project Management Roles

- **Communication Leads:** Hongwei Wang and Perry Ports
- **Scrum Master & Collaboration Lead:** Tabe Ekpombang
- **DevOps & Infrastructure Lead:** Hunter Barton
- **UX/Accessibility Lead:** Hunter Barton

Team Members:

- 1) Adin Huric
- 2) Perry Ports
- 3) Tabé Ekpombang
- 4) Dino Huric
- 5) Hongwei Wang
- 6) Mohammed Abdelgader
- 7) Hunter Barton

Team Procedures

1. **Day, time, and location for regular team meetings:** Weekly meetings in person every Thursday at 2pm.
2. **Preferred method of communication updates, reminders, issues, and scheduling :**
In-person meeting, Discord
3. **Decision-making policy:** Majority vote.
4. **Procedures for record keeping:** Had documents in a shared folder which we used for record keeping

Participation Expectations

1. **Expected individual attendance, punctuality, and participation at all team meetings:** All team members are expected to attend discord meetings unless a circumstance arises. Advanced notice must be provided.
2. **Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:**
Each member is expected to do their assigned tasks on time and communicate if there are any issues in doing so.
3. **Expected level of communication with other team members:** Open communication is required and each team member must provide updates on their progress.
4. **Expected level of commitment to team decisions and tasks:** Team members must respect group decisions to ensure unity.

Leadership

1. Leadership roles:

- Tabe Ekpombang: Individual component design and testing, team organization,
- Adin Huric: Individual component design and testing.
- Dino Huric: Individual component design and testing.
- Mohammed Abdelager: Individual component design and testing.
- Hunter Barton: Individual component design and testing.

- Perry Ports: Individual component design and testing, team organization.
- Hongwei Wang: Client/Advisor Interaction, Individual component design and testing.

2. Strategies for supporting and guiding the work of all team members: Regular check-ins to ensure progress and address any issues or obstacles.

3. Strategies for recognizing the contributions of all team members: Acknowledge individual contributions during meetings as well as using Discord to highlight any milestone achieved by team members.

Collaboration and Inclusion

1. Skills and Expertise:

- Adin Huric: Backend expertise
- Dino Huric: Frontend expertise.
- Tabe Ekpombang: Backend expertise
- Perry Ports: Backend expertise
- Mohammed Abdalgader : Full stack expertise
- Hongwei Wang: Frontend expertise.
- Hunter Barton : Full stack expertise and project innovation.

2. Strategies for encouraging and supporting contributions and ideas from all team members: Foster inclusivity, encourage participation, provide feedback, and celebrate ideas to support team contributions.

3. Procedures for identifying and resolving collaboration or inclusion issues: Goal-Setting, Planning, and Execution.

Goal-Setting, Planning and Execution

1. Team goals for this semester: Fully develop the individual components and integrate to give a fully functional Orgifi.

2. Strategies for planning and assigning individual and team work: Divide tasks based on skills and interests, set clear deadlines, and do regular check-ins.

3. Strategies for keeping on task: Hold regular check-ins, set milestones, and use reminders to ensure accountability and steady progress.

Consequences for Not Adhering to Team Contract

1. **Handling infractions of any of the obligations of this team contract:** Address infractions through open discussion, clarify expectations, and provide an opportunity for improvement.

2. **What will your team do if the infractions continue:** Escalate to involve the instructor or supervisor and reassign responsibilities as needed to ensure project progress.

a. I participated in formulating the standards, roles, and procedures as stated in this contract.

b. I understand that I am obligated to abide by these terms and conditions.

c. I understand that if I do not abide by these terms and conditions I will suffer the consequences as stated in this contract.

consequences as stated in this contract.

1) Adin Huric DATE 05/04/2025

2) Tabe Ekpombang DATE 05/04/2025

3) Perry Ports DATE 05/04/2025

4) Hongwei Wang DATE 05/04/2025

5) Dino Huric DATE 05/04/2025

6) Mohammed Abdelgader DATE 05/04/2025

7) Hunter Barton DATE 05/04/2025