# Tixly

February 5, 2025

# Table of Contents

# Project Proposal and Requirements

## *Introduction*

Tixly is a mobile ticketing application we have designed for our users to discover and manage events and also purchase tickets for those events. The platform will provide access to live entertainment events, including concerts, sports, theater performances, and movies. We aim to create a secure, scalable, and user-friendly system that connects consumers with event organizers while eliminating pain points like ticket fraud, availability and difficult booking processes.

Using tools like cloud computing, microservices architecture, and security frameworks, Tixly ensures real-time availability, personalized recommendations, and a smooth checkout experience. The application is designed to enhance accessibility and engagement, making event attendance more convenient for users while benefiting organizers with a streamlined ticket distribution system.

## *Problem Statement*

Currently, purchasing event tickets online presents several challenges:

- **Fraudulent Ticket Sales:** Consumers fall victim to counterfeit or resold tickets, leading to financial loss.
- **Ambiguous Availability:** Users often struggle to find real-time seat availability, leading to confusion and frustration when attempting to secure tickets for high-demand events.
- **Inefficient Browsing & Discovery:** Existing platforms provide limited recommendation features, making it difficult for users to discover relevant events based on their interests.
- **Complex Booking & Payment Process:** Many ticketing systems involve multiple steps, hidden fees, and poor UI/UX, leading to abandoned transactions.

## *Techniques Used*

We wanted to make sure that the application met user and business needs, so we used the following requirement-gathering techniques:

- **Document Analysis:** We examined existing ticketing applications to identify common challenges and best practices.
- **Comparative Research:** We evaluated competitors' features and weaknesses to improve Tixly's user experience.
- **Stakeholder Discussions:** We identified the needs of event organizers, payment processors, and users to define the platform's features.
- **User-Centric Design Approach:** We followed an iterative design process, starting with low-fidelity wireframes, improving through high-fidelity prototypes, and implementing HCI best practices to optimize usability.

## *Stakeholder Analysis*

Tixly involves multiple stakeholders:

| **Stakeholder** | **Role & Responsibilities** |
|---|---|
| Users | Browse events, purchase tickets, receive notifications, and manage bookings. |
| Event Organizers | List events, set ticket prices, manage seating availability, and track sales. |
| Payment Processors | Secure financial transactions and provide fraud detection. |
| System Administrators | Administer the system so that it remains operational, secure, and scalable. |
| Marketing & Analytics Team | Monitor user behavior to improve recommendations and promotional strategies. |

## *Functional and Non-Functional Requirements*

- **Functional Requirements (System capabilities and features)**
  - User Authentication & Management – Secure login, account creation, and profile management.
  - Event Browsing & Search – Filter events by category, location, date, and popularity.
  - Real-Time Ticket Availability – Users can view up-to-date seat selection and pricing.
  - Secure Ticket Purchase & Payment Processing – Integration with Stripe, PayPal, and Apple Pay for seamless transactions.
  - Fraud Prevention – Unique QR code validation for tickets to prevent unauthorized resales.
  - Personalized Recommendations – AI-driven event suggestions based on past purchases and browsing history.
  - Multi-Channel Notifications – Users receive email, SMS, and push notifications for bookings, promotions, and event reminders.
  - Loyalty & Rewards System – Users earn points for purchases and engagement, redeemable for discounts or VIP perks.
- **Non-Functional Requirements (Performance, security, and system constraints)**
  - Scalability – Cloud-based architecture to handle high traffic spikes during ticket launches.
  - Security & Compliance – End-to-end AES-256 encryption, OAuth authentication, and PCI-DSS compliance for payment security.
  - High Availability & Reliability – 99.9% uptime guarantee with load balancing and failover mechanisms.
  - Performance Optimization – Caching and database indexing to ensure fast event discovery and seat selection.
  - Accessibility – UI with high-contrast visuals, text resizing options, and screen reader support for an inclusive experience.

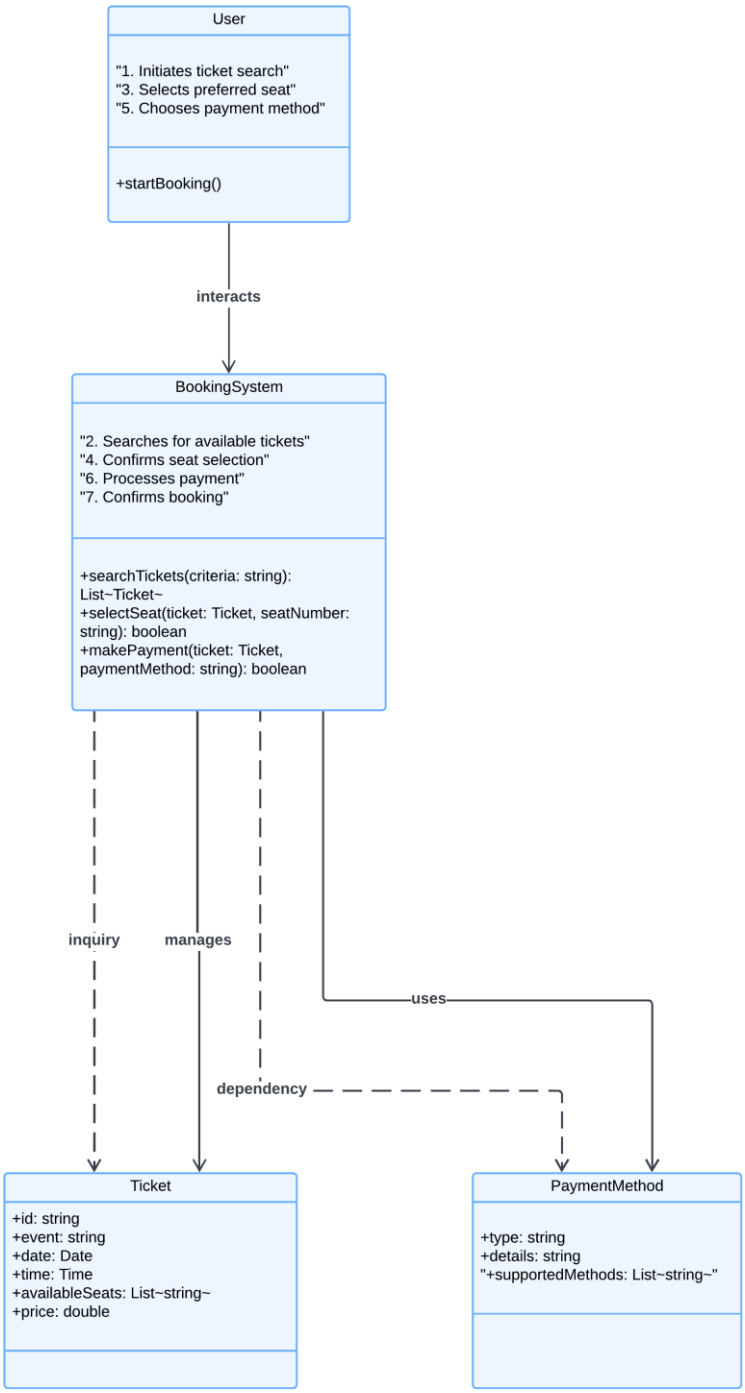


Figure 1: UML Class Diagram of Tixly's System

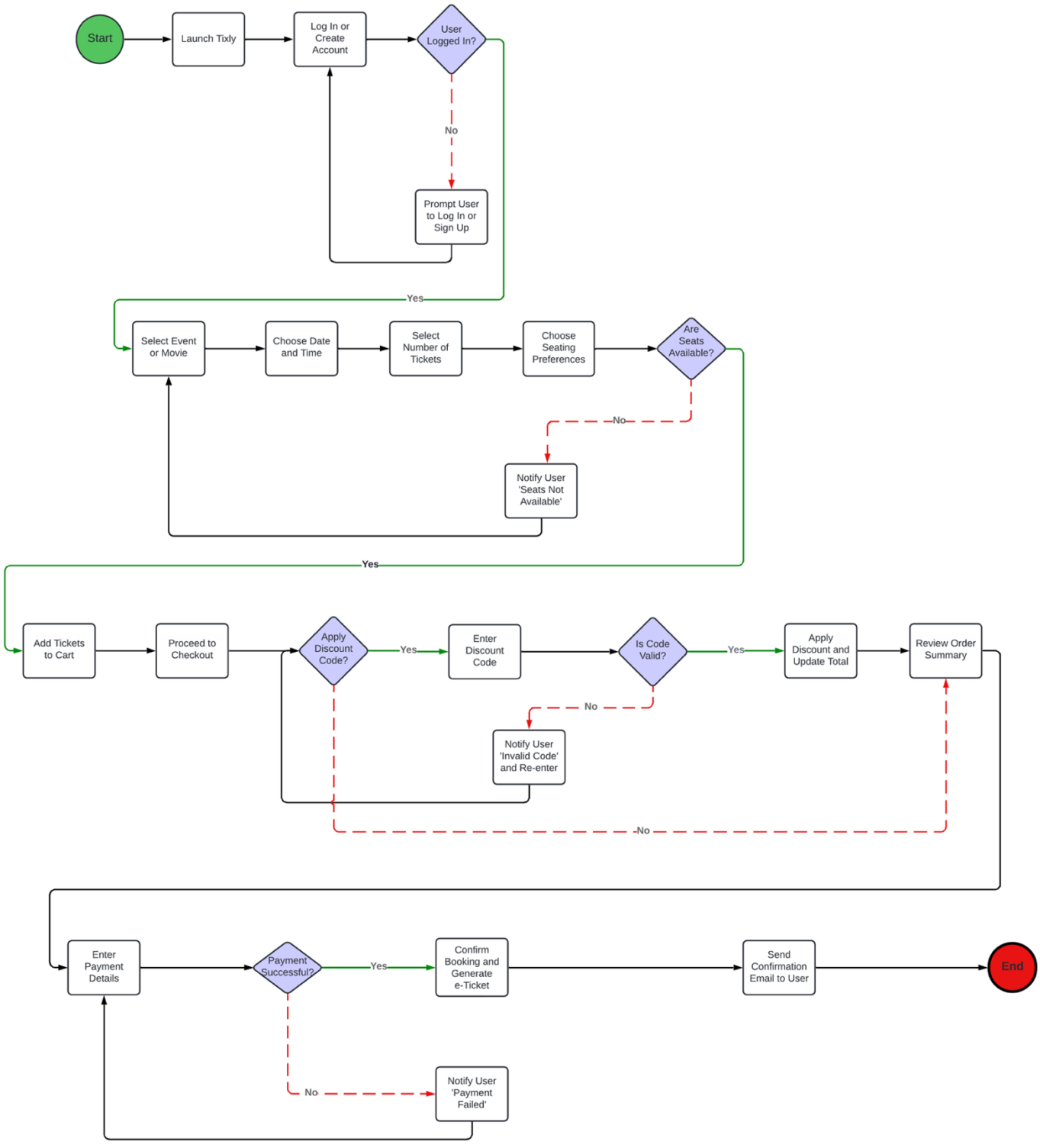


Figure 2: Tixly's Process Flow Chart

# User-Centric Design Document

## *UX Design Principles*

Tixly's User Experience (UX) Design prioritizes user experience, accessibility and ease of interaction to ensure a seamless ticket purchasing process.

- **Consistency**: The interface follows uniform colors, button styles and shapes and also interactive patterns ensuring that there is a level of predictability in navigation for a cohesive experience.
- **Feedback**: This system ensures that users receive instant confirmations, error messages and interactive prompts for critical actions like purchases or filling required fields.
- **Usability**: Complexity is minimized in Tixly by reducing unnecessary steps in event discovery and purchase checkout, allowing users to efficiently complete tasks in the app.

## *UI Design Principles*

The User Interface (UI) design of Tixly focuses on clarity, simplicity and responsiveness to create a visually appealing and intuitive interface.

- **Clarity**: Icons, buttons and labels are clearly defined ensuring that users can easily navigate the app and complete any task without confusion.
- **Simplicity**: Tixly's design avoids clutter by utilizing clean layouts, trivial interactions and minimal distractions to keep things simple for users.
- **Responsiveness**: The app is optimized for different screen sizes and devices, ensuring a consistent experience for users regardless of device.

Integrating these UX and UI principles helps the mobile ticketing app to maximize user satisfaction, enhance usability and ensure a visually engaging experience overall.

## *HCI Best Practices*

We applied Human-Computer Interaction (HCI) best practices to enhance usability and minimize issues in our design. We made sure that our interface followed principles of consistency, clarity, and simplicity, making interactions predictable and efficient.

- **User-Centered Design**: We used internal feedback and insights to do checks on user engagement throughout the design process. This helped us to refine ticket discovery, navigation in the application and checkout flows.
- **Accessibility**: Tixly's interface was designed to accommodate users of all abilities with clear text, high-contrast visuals and intuitive navigation to improve readability and ease of use.
- **Consistency:** We used uniform design elements, including colors, typography, and button styles, across all screens. This provided users with a familiar environment that improved navigation.

### *Prototyping and User Testing*

We followed an iterative approach, moving from low-fidelity wireframes to high-fidelity prototypes in Figma. This allowed us to refine both aesthetics and functionality based on our evaluations.

- **Low-Fidelity Wireframes:** We started with simple sketches to define the layout and flow of the core screens. These wireframes helped us outline functions without focusing on visual details, making sure that the app's structure and navigation were intuitive before improving the design.

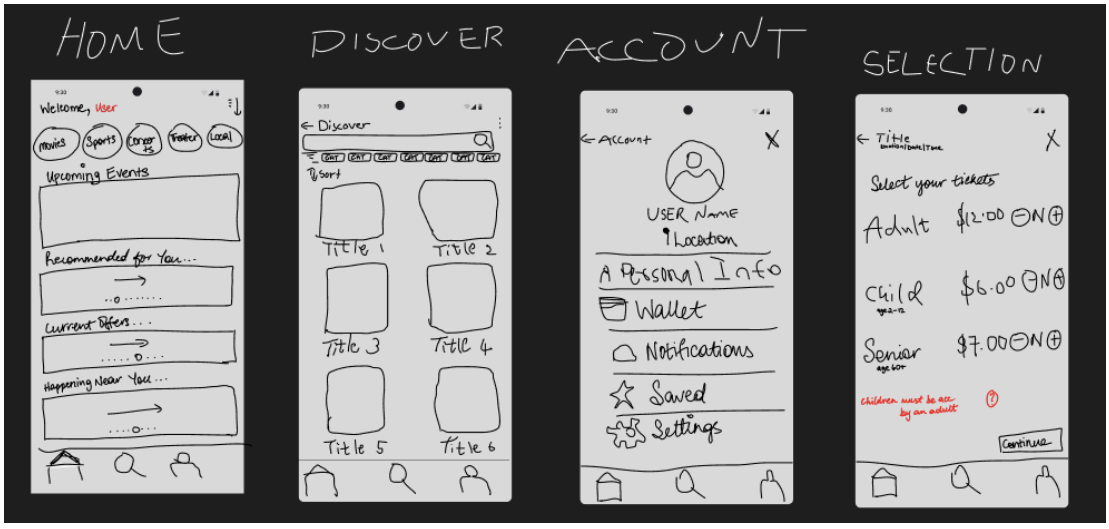


Figure 3: Low Fidelity Wireframes for Tixly's Main Pages

- **High-Fidelity Prototypes:** Once we decided on the layout, we improved on the design with color schemes, typography, and interactive elements to create a user-ready interface. We paid special attention to spacing, alignment, and accessibility to make interactions smooth and enjoyable.

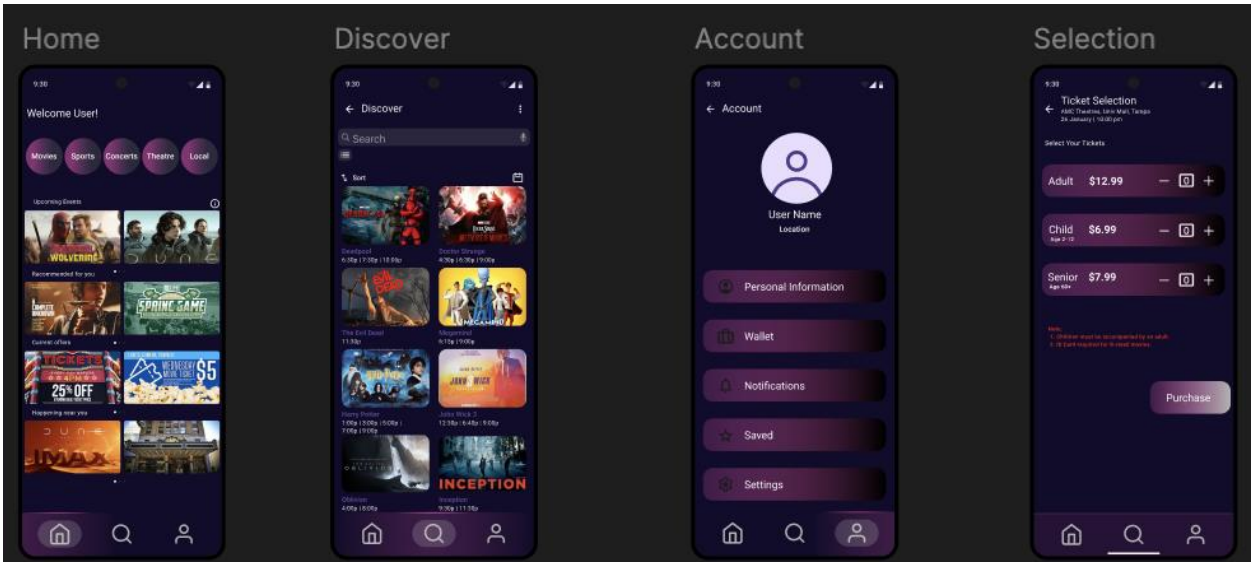


Figure 4: High Fidelity Protypes for Tixly's Main Pages

- **Internal Testing and Refinements:** Without access to external users for usability testing, we conducted internal reviews and walkthroughs. As a team, we simulated user scenarios and interactions to identify areas for improvement. This led to refinements such as adjusting button placement for better reachability, improving text readability, and optimizing navigation paths.

Our iterative prototyping and user testing process helped create a final design that was both visually appealing and functionally intuitive, reinforcing a seamless user experience from discovery to ticket purchase.

# Cloud-Based System Architecture

## *Cloud Service Utilization*

Tixly strategically integrates several cloud services to ensure high system availability, performance, scalability and security. Below is a breakdown of the cloud service utilization across different functionalities.

- **Storage and Database Services**: This system utilizes a combination of relational and non-relational databases to optimize performance, consistency and scalability.

| Service | Database Type | Purpose |
|---|---|---|
| **User Management** | Oracle Database | Structured user authentication data and profile information |
| **Event Management** | Amazon DynamoDB | Enables rapid and scalable storage of dynamic event listings |
| **Booking** | PostgreSQL | Provides ACID compliance for transactional ticket and seat reservations |
| **Payment Processing** | Microsoft SQL Server | Ensures secure financial transactions with robust support |
| **Discover & Recommendation** | Azure Cosmos DB | Facilitates AI-driven event recommendations with global distribution |
| **Review & Rating** | Google Cloud Firestore | Offers real-time synchronization of user-generated content |
| **Notification** | MongoDB | Enables fast access to real-time email, SMS and push notifications |

Scalability Enhancements:

- Sharding and Replication are implemented for distributing data across multiple nodes, ensuring low-latency global access in DynamoDB and Cosmos DB.
- Hybrid Storage: The system uses SQL databases for structured transactional data and NoSQL databases for flexible, event-driven data.
- Load balancing is used across the database services to allow for system availability and scalability by handling traffic spikes efficiently, preventing bottlenecks and optimizing resource utilization. Some of the benefits of load balancing in this system include:
  - Quick retrieval of event listings under high demand in DynamoDB

        - Enhanced personalized event discovery offered by Azure Cosmos DB
        - Instant notification delivery facilitated by MongoDB Atlas
        - Avoiding performance bottlenecks with rating and review submissions in Google Cloud Firestore.

- **Service Communication and Interaction**: Service Communication allows secure, efficient and scalable communication between the system's services, users and external integrations as well.
    - AWS API Gateway manages and secures the API calls between the microservices in the system. This list details the service interactions in Tixly.
        - The User Management Service interacts with
            - ◊ Booking service for booking/purchase history,
            - ◊ Payment Processing Service to retrieve user's payment transaction,
            - ◊ Discover & Recommendation Service for tailored event suggestions.
        - The Event Management Service interacts with
            - ◊ Booking Service to check ticket and seat availability,
            - ◊ Discovery & Recommendation Service to provide event filtering options,
            - ◊ Review & Rating Service to retrieve event reviews.
        - The Booking Service interacts with
            - ◊ User Management Service to fetch/update user's loyalty points,
            - ◊ Event Management Service to get event details,
            - ◊ Payment Processing Service to initiate payment for a booking,
            - ◊ Notification Service to send booking confirmation.
        - The Payment processing Service interacts with
            - ◊ Booking Service to confirm ticket booking upon successful payment,
            - ◊ Notification Service to send payment confirmation/failure notification.
        - Discovery and Recommendation Service interacts with

- ◊ Event Management Service to retrieve event details,
- ◊ Booking Service to use booking history and refine suggestions.

- Review & Rating Service interacts with
  - ◊ User Management Service to authenticate reviews with user profile,
  - ◊ Event Management Service to retrieve event details for review association.

- Asynchronous Messaging – AWS Simple Queue Service
  - The table below describes the asynchronous messaging involved in Tixly's system. These message queues are facilitated by a standard AWS Simple Queue Service (SQS).

    Example: When payments are processed and confirmed, an event is published to a queue that the Notification Service is subscribed to and confirmation email, SMS and push notifications can be sent to users.

| Event | Publisher | Subscriber(s) |
|---|---|---|
| **User Registered** | User Management Service | Notification Service |
| **New Event Created** | Event Management Service | Notification Service, Discovery and Recommendation Service |
| **Booking Confirmed** | Booking Service | Notification Service, Payment Processing Service |
| **Payment Processed** | Payment Processing Service | Notification Service, Booking Service, User Management Service |
| **New Review Submitted** | Review & Rating Service | Event Management Service |
| **Promotion Launched** | Notification Service | Discovery and Recommendation Service |

- **Identity and Access Management (IAM)**: Security for access control is implemented through AWS Identity and Access Management (IAM) frameworks.
    - The AWS IAM Identity Center provides Multi-Factor Authentication (MFA) to strengthen security by requiring additional authentication layers for user login and admin access.
    - AWS Key Management System (KMS) allows data encryption to keep user credentials, payment details and sensitive information encrypted and hence protected at rest and in transit.
    - Role-Based Access Control (RBAC) ensures least privilege access to restrict database queries and administrative privileges. AWS allows this with the use of IAM roles, users and policies.
    - Amazon Cognito provides JSON Web Tokens authentication for API Gateway to secure the API requests and prevent unauthorized access to the microservices in Tixly.

Tixly leverages AWS, Azure and Google Cloud Services to achieve high performance and scalability with good security and efficiency. The cloud architecture integrates storage, database services, load balancing, asynchronous messaging and identity management to form an optimal cloud-based architecture. The application delivers a resilient, efficient and seamless ticketing experience for users.

### *Microservices Architecture*

The microservices architecture implemented in Tixly ensures that the system is scalable, maintainable and resilient, which allows independent service scaling and continuous deployment. Breaking the system into specialized microservices allows each service to operate independently to allow for auto-scaling and load balancing based on demand to optimize performance during high traffic like event launches or ticket releases.

Asynchronous messaging queues help to ensure there is proper event processing which prevents system slowdowns while handling notifications and transactions efficiently. Services communicatee via API gateways to ensure seamless interaction while enforcing security and authentication.

For scalability, the application's architecture supports independent and horizontal scaling allowing high-demand services to expand dynamically and optimize resource efficiency therein contributing to system performance and availability.

With regards to maintainability, microservices allow for modular deployments which helps to reduce downtime and allow for rapid feature updates in the system. Although the microservices architecture tends to introduce complexities, Tixly's architecture ensures resilience, scalability and maintainability which makes it a high-performing ticketing application.

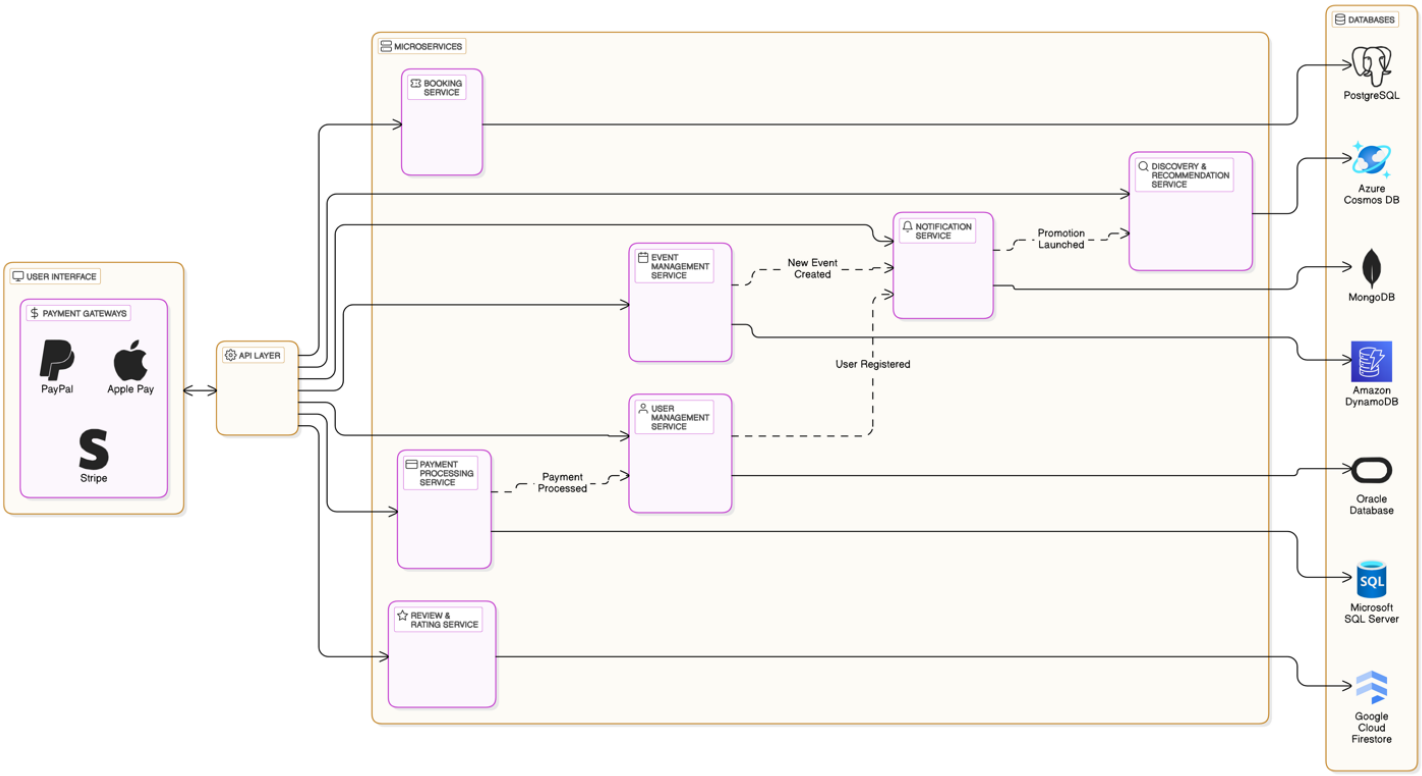


Figure 5: Microservices Architecture for Tixly

# Agile Project Delivery and Management Strategy

## *Methodology Application*

Our team decided that the Kanban framework would work best for our project as it allowed us to visualize our workflow, prioritize tasks, and continuously improve our system. Using the Agile principles, we focused on iterative development and strong collaboration to improve on several components of our project, from system requirements to UI/UX design and cloud architecture.

We decided on Kanban as our Agile framework due to its flexibility and ability to manage workflow efficiently. Kanban allowed us to structure our project into clearly defined tasks while maintaining adaptability. The key aspects we applied included:

- **Workflow Visualization:** We tracked our progress using a Kanban board, ensuring a clear view of completed, ongoing, and pending tasks.
- **Task Prioritization:** We focused first on core functionalities, such as ticket browsing and purchasing, before expanding to additional features.
- **Continuous Improvement:** Our workflow allowed us to revisit and refine system requirements, UI/UX elements, and overall design as new ideas emerged.

## *Iterative Development*

Iterative development played a crucial role in shaping the project. Through multiple iterations, we improved various aspects of the system:

- **UI/UX Refinement:** We transitioned from low-fidelity wireframes to high-fidelity prototypes in Figma, making sure we provided a seamless user experience by improving clarity, usability, and accessibility in each iteration.
- **System Architecture Enhancements:** We continuously refined our microservices architecture, database structures, and cloud service integrations to ensure scalability and security.
- **Requirement Adjustments:** As we progressed, we identified and adjusted system requirements to align with real-world ticketing needs, ensuring the project's feasibility and effectiveness.

### *Team Collaboration*

Collaboration was important to successfully managing our project. We used the following strategies to stay productive:

- **Kanban Boards:** We used Kanban boards to organize tasks, monitor progress, and ensure efficient delegation.
- **Regular Discussions:** Frequent team meetings allowed us to share insights, discuss improvements, and resolve challenges effectively.
- **Documentation Alignment:** We worked together to stay consistent across all project documentation, from requirements gathering to system architecture.
- **Feedback Integration:** Maintaining an open line of communication allowed for significant improvements in design, security considerations, and overall project structure.

By following Agile principles and utilizing Kanban, we remained organized and adaptable. Our iterative development process promoted thoughtful progress, leading to a well-structured and comprehensive final deliverable.

# Technology Stack, Security, and Scalability Strategy

## *Technology Selection*

The Tixly app has been designed to be scalable, secure, and efficient. It leverages various modern technologies to support real-time operations. The choice of technologies is based on the requirements of the system, such as high availability, security and seamless user experience.

- **Frontend:** We selected React Native to build the mobile application due to its cross-platform capabilities. It allowed us to develop the application for both iOS and Android using a single codebase, thereby ensuring consistent user experience across devices.
- **Backend:** When it came to the backend, we went ahead with Node.js as its non-blocking, event-driven architecture is ideal for handling multiple connections simultaneously and effectively. This is particularly crucial for features like ticket availability checks.
- **Database:**

| Service | Database Type | Reasoning |
|---|---|---|
| **User Management Service** | Oracle Database | Stores structured user profiles, authentication credentials, and loyalty points with relational integrity. Oracle Database offers high security, making it ideal for sensitive user credentials. |
| **Event Management Service** | Amazon DynamoDB | Manages dynamic event listings, descriptions, images, pricing, and metadata in a flexible document-based format. DynamoDB provides fast and predictable performance with seamless scalability. |
| **Booking Service** | PostgreSQL | Ensures transactional consistency for ticket reservations, seat selection, and booking confirmations. PostgreSQL offers robust ACID compliance and advanced features for complex queries. |

| | | |
|---|---|---|
| **Payment Processing Service** | Microsoft SQL Server | Secure financial transactions require ACID compliance to prevent inconsistencies. SQL Server provides strong transactional support and integration with other Microsoft services. |
| **Discover & Recommendation Service** | Azure Cosmos DB | Provides AI-driven event recommendations and search indexing for better scalability. Cosmos DB offers global distribution and multi-model support, making it ideal for scalable search and recommendation systems. |
| **Review & Rating Service** | Google Cloud Firestore | Handles user-generated reviews and ratings efficiently, allowing flexible data storage. Firestore supports real-time synchronization and is optimized for mobile and web applications. |
| **Notification Service** | MongoDB | Stores real-time messages, push notifications, and email/SMS logs for fast access. MongoDB's flexible schema design is ideal for handling diverse notification formats. |

- **Payment Gateway:** The application integrates payment services such as Stripe, PayPal, Apple Pay, Google Pay etc. to provide secure, fast and seamless payment transactions. These gateways provide robust security features, ease of integration and have a global reach. All the transactions comply with PCI-DSS standards (Payment Card Industry Data Security Standard), ensuring end-to-end encryption, tokenization, and fraud prevention mechanisms. By employing AI-powered fraud detection, 3D Secure authentication(3DS2), and real-time risk assessment, we prevent unauthorized purchases in the application.
  Additionally, the platform supports multi-currency payments, mobile-friendly checkout experiences (Apple Pay & Google Pay), and one-click transactions, enhancing user convenience. With chargeback protection and dispute resolution, users and event organizers are safeguarded against fraudulent transactions.

### *Security Framework*

Security is a top priority in any ticketing platform, and Tixly follows zero-trust architecture with multi-layered security implementations to prevent fraud, data breaches and unauthorized access. Security features include:

- **User Authentication & Authorization**: By implementing OAuth 2.0 for secure user authentication and authorization, we ensure that certain features and services can only be accessed by authenticated users.
    - Implemented OAuth 2.0 with JWT (JSON Web Tokens) for secure API authentication.
    - The application supports Multi-Factor Authentication (MFA) for account security.
- **Data Encryption**: All the sensitive data in the application is encrypted using industry-standard protocols.
    - AES-256 encryption is used for data at rest. This protects sensitive user information such as passwords and payment details.
    - TLS 1.3 is used for data in transit. This encryption ensures secure data transmission between the client and server.
- **Fraud Prevention & Threat Mitigation**: We conduct regular security audits and penetration testing to identify and mitigate any potential vulnerabilities in the application. We also use Web Application Firewalls (WAF) to protect against common web exploits and attacks.
    - AI-Powered Fraud Detection: This is used to identify anomalies in purchasing behavior (e.g., multiple purchases from a single IP).
    - Blockchain-Based Ticket Verification: By assigning a unique digital signature to each ticket, we prevent resale fraud.
    - DDoS Protection: We use Cloudflare to protect the application against distributed denial-of-service attacks.
- **Compliance & Regulations**: Tixly strictly adheres to compliances such as the General Data Protection Regulation (GDPR) to safeguard user data. The highest standards of

privacy and security are maintained in data collection, processing and storage. With transparent policies regarding data usage and consent management, users have full control over their information and how it is used.

All payment transactions comply with Payment Card Industry Data Security Standards (PCI-DSS), a globally recognized framework designed to protect financial data. This ensures that every transaction is securely encrypted, reducing the risk of fraud and unauthorized access.

### *Scalability Measures*

The application has been designed to handle millions of concurrent users without a drop in performance. Scalability strategies include:

- **Load Balancing**: We use cloud-native load balancing services such as AWS Elastic Load Balancing (ELB), Azure Load Balancer, or Google Cloud Load Balancing to distribute incoming traffic across multiple instances of our application, ensuring high availability and reliability
- **Caching for Performance Optimization**: Services like Amazon ElastiCache, Azure Cache for Redis, or Google Cloud Memorystore have been used to implement caching mechanisms to store frequently accessed data. This helps reduce the load on the database and improve response times.
    - Redis is implemented to cache frequently accessed data (e.g., event listings, ticket availability) to reduce database load.
    - CDN (Content Delivery Network) ensures faster content retrieval for users across different regions.
- **Database Sharding & Replication**: To enable the system to handle large volumes of transactions and queries efficiently, we implemented database sharding to distribute data across multiple instances. This is crucial in services like the Booking Service, which expects millions of concurrent transactions.
    - Horizontal Scaling: PostgreSQL and MongoDB are sharded to distribute data across multiple databases.

  - Read Replicas: Uses multiple database instances to offload read operations, ensuring faster queries.

- **Auto-Scaling & Cloud-Based Deployment**: Tixly is built for efficiency and scalability, ensuring smooth performance even during high-demand events. By using Docker containers with Kubernetes, the system can automatically scale up or down based on traffic, preventing slowdowns or crashes.

  Additionally, serverless computing with AWS Lambda handles critical event-driven tasks like sending notifications and verifying tickets, ensuring fast, reliable processing without the need for constant server management. This approach keeps the system cost-effective, highly available, and responsive to user needs.