tixar.sg

# TIXAR

14/11/2023

**Prepared For :**
Shao Jianhua
Lydia Hamid

**Singapore Management University**
CS203 Software Development Methodology

## Table of Contents

# 1. Overview

[Tixar's Mobile Git Repository]
[Tixar's Verified Fans Git Repository]
[Tixar's Reliable Ticketing Git Repository]

**Background**

"Taylor Swift fans outraged over Ticketmaster system issues as UOB pre-sale concert tickets snapped up in 3 hours".

**Theme: "Fair and reliable ticketing system".**

Our team's answer to this is Tixar, an innovative ticketing platform that provides solutions to the problems that arose from Ticketmaster's ticket sales. To address the problems, Tixar's features are:

Fair Ticketing

Verified Fan Point System: We introduce a priority ticketing mechanism that rewards dedicated fans. Our artists will periodically distribute unique codes. Fans who collect these codes demonstrate their loyalty and are more likely to be true supporters. Priority in ticketing will be given to these users, ensuring that true fans have better access to event tickets.

Combatting Scalpers with Unique Identification: To discourage scalping, our system requires a verified phone number for account login. This measure significantly increases the difficulty of creating and using multiple accounts for the purpose of ticket scalping. By linking each account to a unique phone number, we ensure a fairer ticket distribution and prevent bulk buying by scalpers.

Reliable Ticketing

Staggered Ticket Release: We will implement a distributed release of tickets in waves. This approach helps in managing server load more effectively by limiting the number of users who can access ticket sales at any given time. Access to these sales waves will be granted based on accumulated Verified Fan Points or special promotional criteria, ensuring a smooth and orderly ticket purchasing experience.

Integrated eWallet Feature: To further streamline the ticket purchasing process and reduce congestion on the transaction gateway, we introduce an eWallet feature. This functionality allows users to preload funds into their account wallet, enabling quicker transactions and alleviating the traffic during each wave of sales. This feature not only speeds up the checkout process but also enhances user experience by minimising transaction-related delays.

# 2. Software Process

**Methodology**

Understanding the short timeline that our team has to develop Tixar, we have adopted a Scrum Agile Methodology, which will provide the foundational framework for our team to focus on productivity, flexibility, improved collaboration and faster time-to-market.

**Sprint 1**

We began on 20th September 2023 and ended on the 25th of September 2023. Before Sprint 1 began, our team held a sprint meeting to kick off sprint 1. By the end of the meeting, our team agreed to include the following user stories in our sprint, and assigned story points to each story through planning poker:

- (IAM) Features: (Login, Profile)
- (FAN) Features: (Code Redemption, Fan Dashboard)
- (CORE) Project Initialization: Technical Stories

Our team's aim in sprint 1 was the completion of stories from 3 epics: IAM, FAN and CORE, totaling 48 story points (fig 2.1).

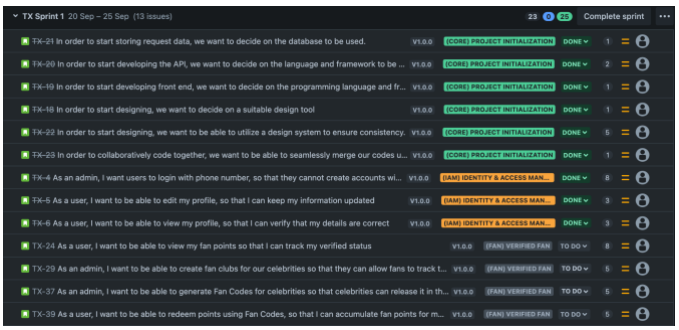


*Fig 2.1: User stories for sprint 1*

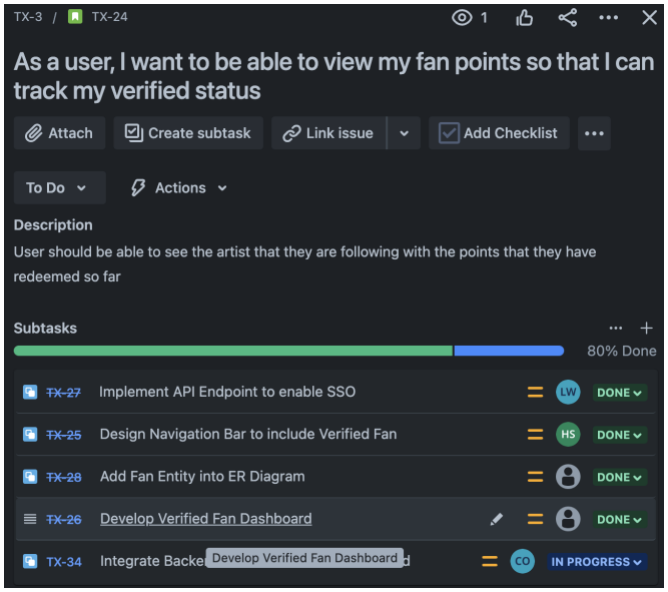


*Fig 2.2: User story details*

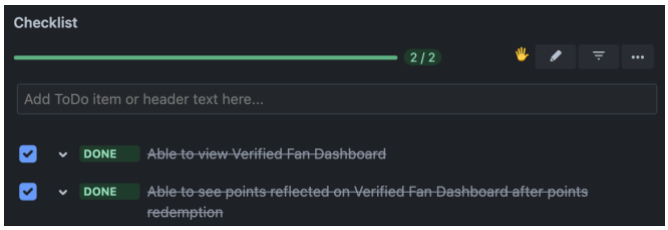


*Fig 2.3: User story definition of done (DOD)*

Each Epic consists of multiple user stories, each of which represents a feature our team wanted to implement. For example, in IAM, the above (Fig 2.2) was a user story scheduled to be completed in Sprint 1, representing the login feature that we wanted to implement. The user story is structured so that the feature is fully developed before being packaged and deployed. In Figure 2.2 we can see a description, which describes the end product of what a user should be able to do with the login feature. The user story is then given subtasks in the following categories:

- Design - Design UI wireframe on Figma
- Develop - Develop front-end UI with React Native
- Implement - Implement back-end API with express to handle login requests
- Integrate - Connect front-end and back-end to complete the login feature

The user story is then given a Definition of Done (DOD) (Fig 2.3) as a checklist of acceptance criteria at the bottom. In this case, before the points tracking feature is considered completed, users should be able to view their fan dashboard and the respective points. Based on the subtasks and DOD, the user story is then given a story point to estimate the effort required for this user story.

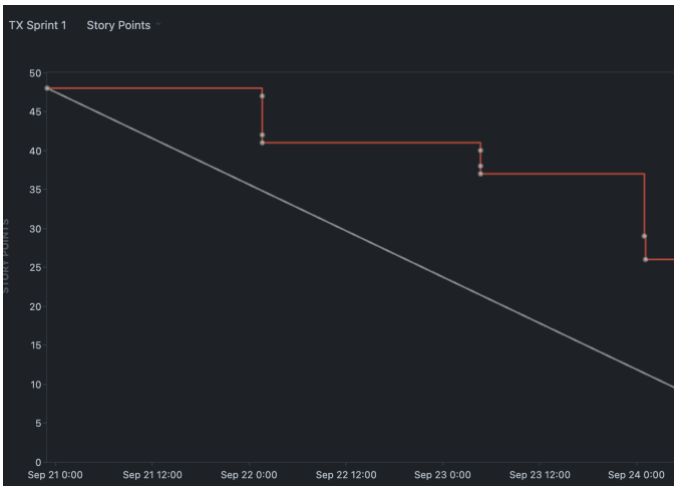


*Fig 2.4: Burndown chart for Sprint 1*

### Progress Tracking

We tracked our team's progress using a burndown chart (Fig 2.4). The burndown chart allows us to see our team's sprint velocity during our sprint. Every day, our team did a daily standup to check up on every individual's progress and find out if there were any impediments, allowing us to quickly identify any roadblocks and provide a solution to remedy them as a team. In Figure 2.4, the backlog for Sprint 1 decreased from 48 to 37 as the technical stories for Tixar were completed and documented. Once the technical foundation was laid down, our team could then start the development of Tixar.

### Sprint Retrospective

Upon closing the sprint, the team conducted a sprint retrospective to come together and close off the sprint and reflect on what we have done so far. During the sprint, the team did well communicating with each other on impediments which allowed us to progress through our sprint. We also noted that we could ensure that we standardised pages as we had issues locating and navigating to the correct pages. In our first sprint between weeks 3 and 4, we closed off 1 sprint with a retrospective which unveiled inconsistencies in how we applied agile methodology. This allowed us to correct our mistakes by the 2nd sprint quickly.

## Sprint 2


*Fig 2.5: Burndown chart for Sprint 2*

Sprint 2's duration was from the 3rd of October to the 8th of October. The Sprint started after the first V.1.0.0 demo of the app. During the Sprint Meeting, we identified another area we could improve on and that was on the team's role in development. Originally, the team had 5 front-end developers and 1 back-end developer. This meant that the front end was generated quickly. Because of this, there was a lot of waiting that happened. After identifying the issue, we decided to shift the responsibilities and came up with the current roles and responsibilities according to the team's strengths.

### Roles and Responsibilities

| | |
|---|---|
| Lucas Liao | Product Owner<br>Backend Developer<br>Devops |
| Haris Shariff | Scrum Master<br>Backend Developer |
| Lim Wei Jie | Frontend Developer<br>UI Designer |
| Oh Sheow Woon | Frontend Developer |
| Chloe Ong | Integration |
| Khoo Jing Ren | Integration |

With the reorganisation of the team, the team was ready to perform optimally according to our strengths.

For Sprint 2, we focused on bug fixes and refactoring of the demonstrated version over 32 points worth of User Stories that covered the entirety of the Verified Fan Epic.

### Sprint Retrospective

During the Sprint Retrospective, we noted the team's overall efficiency with the change in roles. As seen in the above screenshot (Fig 2.5), the team performed efficiently completing all 32 points within the stipulated time. Our daily standups further supported the team's workflow as we were able to redirect any impediments to the right team. For example, if there were any backend impediments, there were now 2 people who were able to assist. Lastly, due to server costs, the team decided to reconfigure our backend architecture which will be covered in the technical section.

### Sprint 3 - Final Sprint

During the upcoming weeks leading up to Sprint 3, there was a big block in the team's progress due to quizzes and assignments from other courses. However, the team finally was able to come together for the final Sprint Meeting to discuss the final feature that we were going to implement. The team pushed in the final 26 story points consisting of the final epic to be completed - Concert Ticketing. The goal of the final sprint was to complete the following features:

- Available Concerts
- Concert Selection and Information
- Transaction features
- eWallet Payment feature

Sprint 3 began on the 2nd of November and ended on the 6th of November. As with the previous sprint, the team was still very efficient as by Sprint 3 all group dynamics and impediments were solved.


*Fig 2.6: Burndown chart for Sprint 3*

The final sprint also consisted of tidying up all bugs, refactoring and preparing for the final demo.

## 3. Technical Details

In this section of the report, not only will we introduce the technologies utilised for this project but also cover key decisions made and iterative progression on how we have shifted the adoption of technologies throughout the journey of developing Tixar.

In crafting a scalable and efficient digital ecosystem, it is crucial to understand not only the final architecture but also the evolution of the technology stack and the rationale behind each transition. We will start by diving deeper into the initial considerations. Subsequently, we will touch on modifications, and the final choices in our tech and deployment stack, providing insights into the dynamic nature of technology decision-making.

**Initial Technology Stack**

Express.js and Java Spring Boot

Initially, Tixar's ticketing system was built using Java Spring Boot, renowned for its robustness and multi-threading capabilities. However, as the project evolved, particularly with the MVP (Minimum Viable Product) in mind, a pivot was made towards Express.js. This decision was driven by the need for rapid development and deployment. Express.js, with its minimalistic yet powerful framework, allowed for quicker iterations, which is essential in an MVP approach.

Heroku and ObjectRocket

In the beginning stages, Heroku was chosen as the host provider for its simplicity in deployment and ease of use. It allowed our team to focus on development without worrying about the complexities of server management. For databases, ObjectRocket was used for MongoDB hosting. which offered a managed database service. However, as the project scaled, the costs associated with ObjectRocket and the limitations of Heroku in terms of cloud deployment strategies became apparent.

**Transition to AWS and Refined Stack with Kubernetes**

As the project grew, the need for a more scalable, customizable, and cost-efficient solution became evident. This led to the transition to AWS, a more robust and scalable cloud service provider. AWS offered a wider range of services that were essential for scaling and managing our application effectively.

**AWS EC2**: Provided scalable server infrastructure.
**AWS S3**: For reliable and scalable cloud storage solutions.
**AWS DynamoDB**: Was initially considered for its highly scalable and flexible NoSQL database service.
**AWS EKS**: Recognizing the need for advanced orchestration and container management for the microservice architecture utilised in Tixar, we incorporated AWS EKS and Kubernetes into our architecture. Kubernetes provided the tools necessary for automating deployment, scaling, and operation of application containers across clusters of hosts. AWS EKS offers a managed Kubernetes service, simplifying the process of running Kubernetes on AWS without needing to install and operate your own Kubernetes control plane or nodes. This integration brought enhanced scalability, reliability, and efficiency to our deployment processes.

However, the team soon realised that the service costs were exceeding the AWS free tier, leading to higher operational costs.

**Final Technology and Deployment Stack**

With these considerations in mind, we deliberated as a team and came to utilise the following technology stack and architecture for the final version of our product.

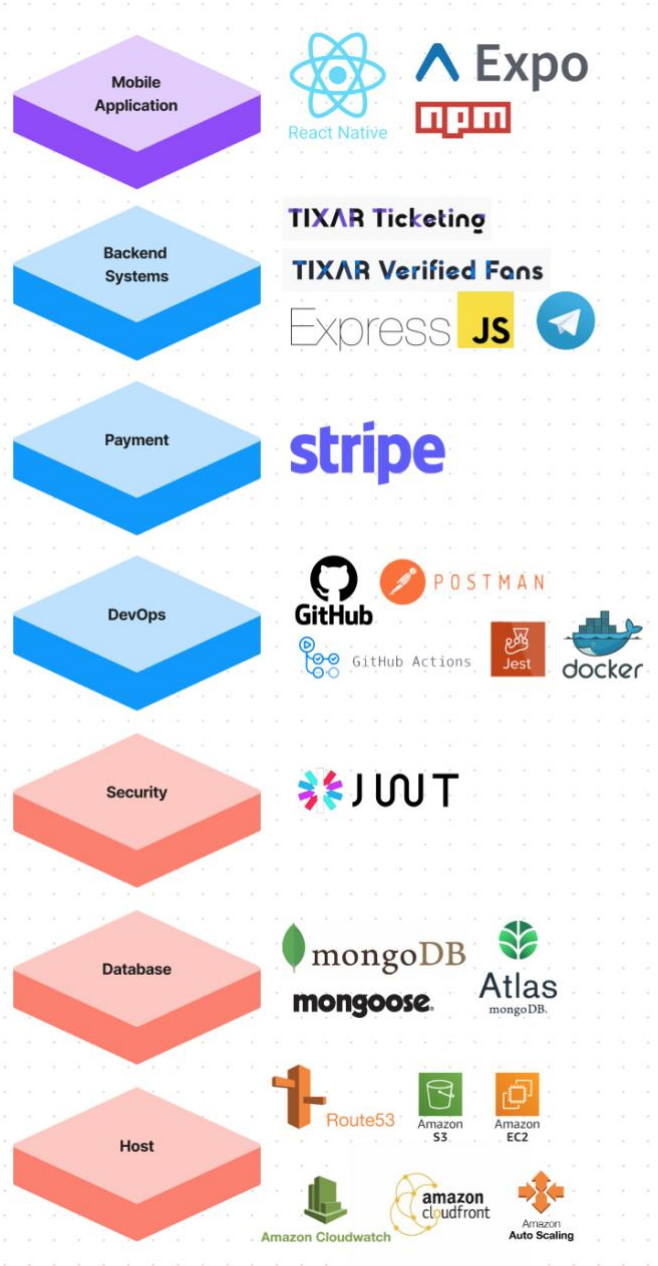


*Fig 3.1: Tixar's technology stack overview*

**Mobile Application**

React Native

The cornerstone of user interaction in our project is the mobile application, developed using React Native. This choice brings the power of native performance coupled with the flexibility of JavaScript, allowing for a seamless user experience across both iOS and Android platforms. React Native's component-based architecture enables fast development and easy maintenance, making it an ideal choice for modern mobile applications.

The decision to use React Native remained constant due to its cross-platform capabilities and efficient development cycle.

**Backend Systems**

Express.js for Core Systems

At the heart of our backend infrastructure are three critical systems: the Ticketing System, Verified Fans System, and the Telegram OTP System, all developed using Express.js. This minimalist web framework for Node.js offers a robust set of features for web and mobile applications, making it a perfect fit for building fast, scalable network applications.

Ticketing System

This system is the backbone of our service, handling all aspects of ticketing transactions. Its design, hosted on AWS, ensures scalability to handle high volumes of traffic, especially during peak sales periods.

Verified Fans System

This system is the unique feature of our product, which ensures fair distribution of tickets using a points system, helping to ensure equity in ticketing.

Telegram OTP System

Integrated with the Telegram API, this system provides a secure and user-friendly OTP verification process, enhancing the security and usability of our application.

The shift from Java Spring Boot to Express.js for the Ticketing System was crucial to align with the rapid development needs of the MVP.

**Payment Processing**

For financial transactions, the integration of the Stripe API is pivotal. Stripe's robust infrastructure for handling online payments ensures secure and efficient processing of transactions, which is crucial in maintaining user trust and compliance with financial regulations.

**Database**

MongoDB and Mongoose

Our database choice is MongoDB, a NoSQL database that offers scalability and flexibility in

handling large volumes of data. Mongoose, a MongoDB object modelling tool, provides a straightforward, schema-based solution to model our application data. MongoDB Atlas VPC on AWS EC2 hosts our database, offering enhanced security and scalability.

MongoDB, initially hosted on ObjectRocket, was shifted to MongoDB Atlas on EC2. This move offered better integration with AWS services and improved cost-effectiveness.

**Development and Operations**

GitHub

Used for version control, GitHub allowed us to manage and track changes in the software development process efficiently.

Postman

We utilised this API platform to build and test our APIs. This ensured that our APIs were robust, well-documented, and easy to use.

Jest

We used this JavaScript Testing Framework, which has a focus on simplicity, providing a robust foundation for the test-driven development (TDD) approach that our team adopted, utilising regression testing.

GitHub Actions for CI/CD

This automated our software deployment pipeline, reducing the possibility of human error on our side and increased deployment efficiency.

Docker

This ensured consistency across various development and production environments, solving the 'it works on my machine' problem.

**Security Measures**

In terms of security, our architecture employs JWT tokens for secure user authentication, as well as a custom-scripted function for IP blacklisting, thereby enhancing the security of our systems. Additionally, express-rate-limiter is used to prevent abuse of our services, ensuring that our systems are resilient against brute-force and DDoS attacks.

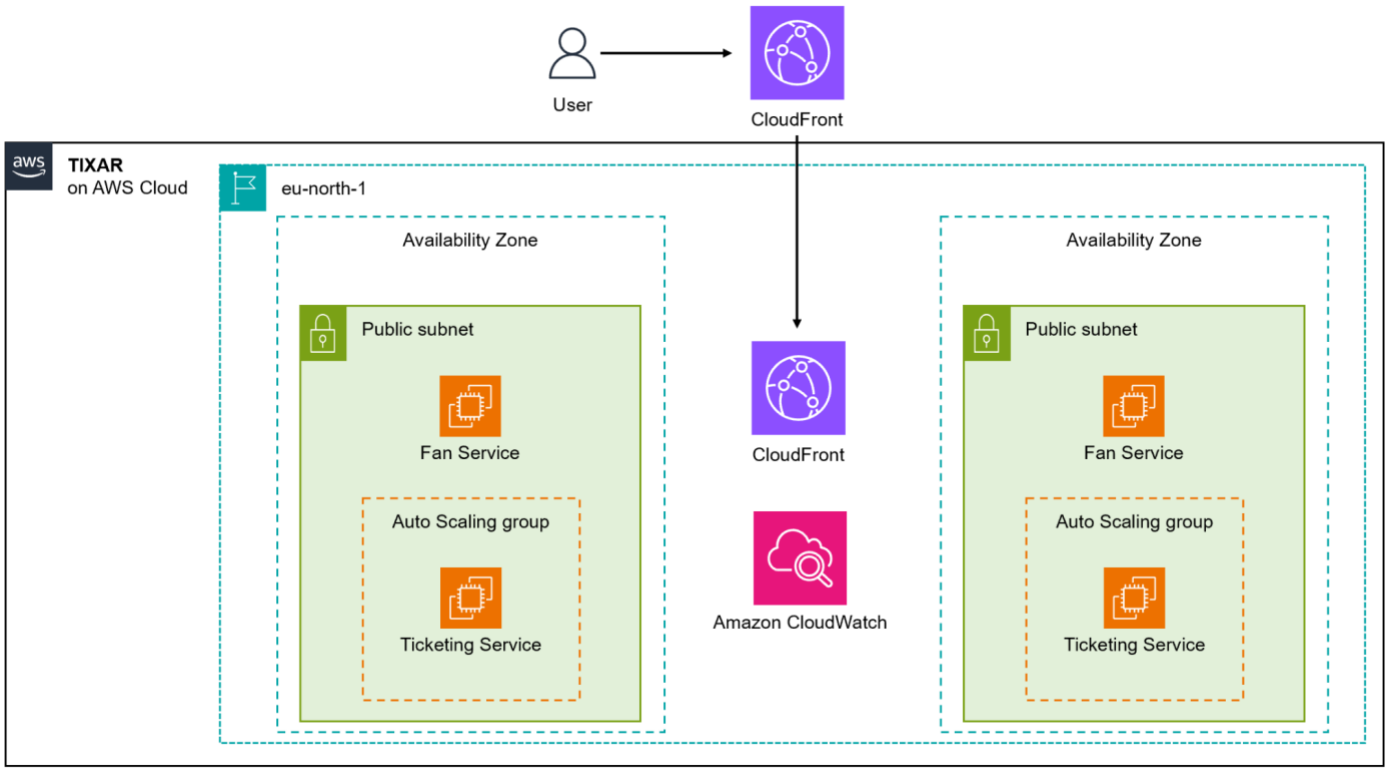


*Fig 3.2: Hosting and cloud services architecture*

AWS Route53
This manages the DNS, ensuring that our web applications are easily accessible and reliably routed.

AWS S3 Bucket
This stores images and static assets, providing a highly durable, scalable, and secure solution for any storage required.

AWS EC2
This hosts our applications, offering scalable server infrastructure to handle varying loads.

AWS Auto Scale
This dynamically adjusts the resources based on traffic and demand, ensuring that our application remains responsive during peak loads, such as during ticket sales.

AWS Availability Zones
This offers high availability and fault tolerance to our services.

AWS CloudWatch
This provides monitoring services for our application, allowing us to easily collect and access performance and operational data.

AWS CloudFront
This web service speeds up the distribution of static and dynamic web content to users of our application.

The evolution of our tech and deployment stack from its initial conception to its final form reflects a journey of adapting to changing requirements, optimising costs, and embracing the best technologies for our purposes. Each transition, from the shift to Express.js for faster development cycles to the move to AWS for better scalability and cost management, was driven by the project's growing needs and the evolving landscape of technology solutions. This journey underscores the importance of flexibility, foresight, and a keen understanding of technological capabilities in building a robust, scalable, and efficient digital platform.

# 4. Database Design

**Overview**
Our database is meticulously designed to support our intended features. We employed a relational model (Fig 4.1) to ensure data integrity and facilitate complex queries that are essential for such a multifaceted platform.

**Design Details**
User-Centric Design
The USER table is central to the design, with fields to capture essential user details, including contact information and encrypted authentication data like otpValue. This user-centric approach enables personalised experiences and secure interactions within the platform.

Role-Based Access
The ADMIN entity allows for role-based access control, indicating a clear separation between regular users and administrators. This distinction is crucial for managing permissions and maintaining platform security.

Fan Engagement
The FAN, PROFILE, CLUB, and GOODIES entities are interconnected, representing the verified fan instance. This design encourages user engagement by tracking fan loyalty and enabling the redemption of points for goods, highlighting a gamified aspect of the user experience.

Event Management
The EVENT, SALESROUND, SESSION, PRICE, and CAPACITY entities are related to event management. This allows for detailed tracking of events, their sales rounds, and session times, along with managing ticket pricing and venue capacity. Such granularity ensures flexibility in event setup and ticket sales strategy.

Transactional Integrity
The TRANSACTION entity, linked to both USER and TICKET, captures the details of ticket purchases. The use of a separate CODE entity for promotional or verification codes ensures that the

system can handle various types of transactions securely and efficiently.

Payment Handling

The CUSTOMER table is associated with the USER table, including payment details and eWallet balances. This separation of concerns indicates a focus on secure payment processing, which is essential for financial transactions.

**Design Objectives**

Scalability

The normalised form of the database with well-defined relationships allows for scalability. As the platform grows, new entities and relations can be added without major overhauls.

Flexibility

The modular nature of the event-related tables provides the flexibility to handle various types of events, pricing models, and venue sizes, catering to a wide range of event management scenarios.

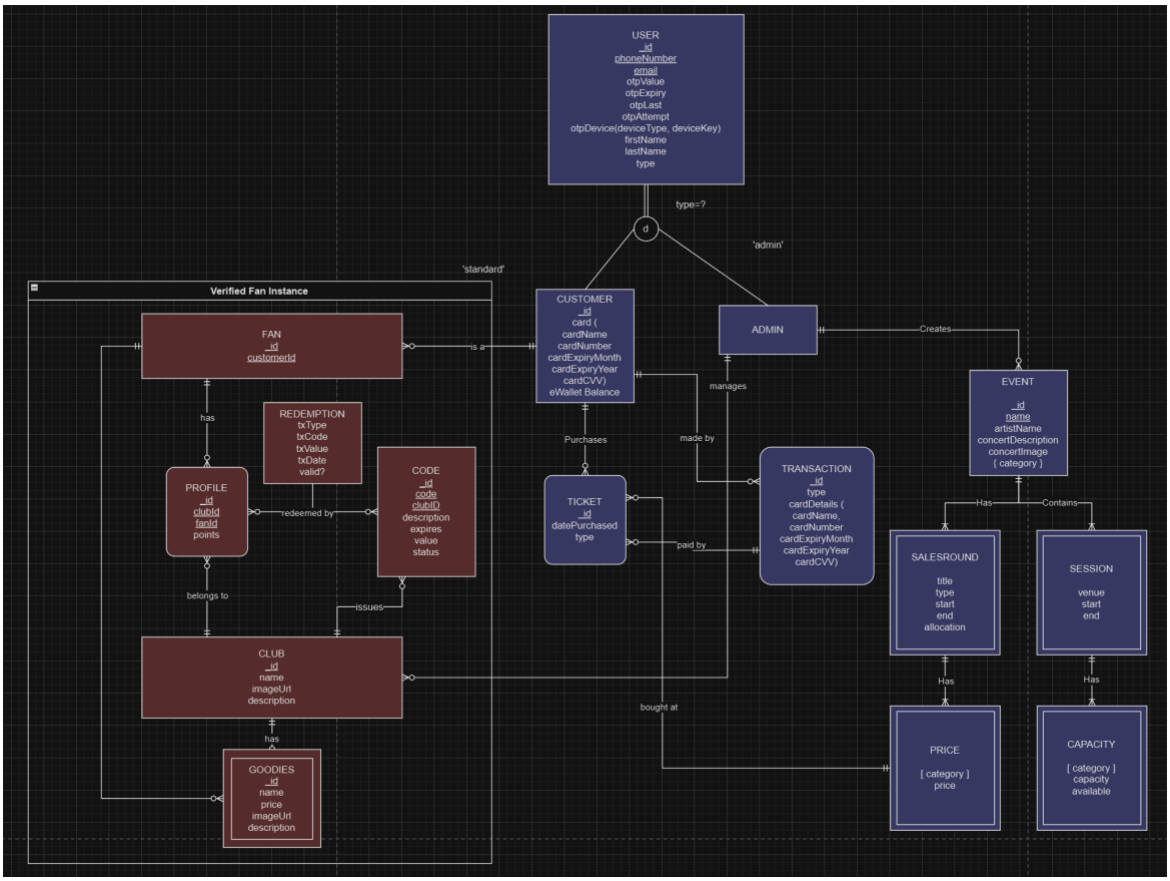


*Fig 4.1: Database schema*

## 5. Conclusion

Tixar represents a significant leap forward in creating a fan-centric ticketing ecosystem. With the introduction of our Verified Fan ticketing system, we've opened a new realm where genuine fans are prioritised, enhancing their chances of engaging with their favourite celebrities.

Our system empowers true fans, those who diligently follow their favourite artists across various platforms like Instagram, fan meets, music videos, podcasts, and more, by offering them exclusive access to tickets through unique codes. These codes are more than just passkeys; they're a bridge connecting fans to memorable experiences.

In essence, Tixar effectively tackles our target issue of fair and reliable ticketing, providing a gateway to unforgettable encounters for devoted fans.