



# ProgChamp

Week 0

## Mentors:

Rehaan Jose Mathew

Ashmita Chaki

Zia Kadijah

Mebin Thattil

## Mentees:

Archita Agrawal

Sannidhi Nayak

Poornaprajna Kashyap

Pratham Mudakavi

Syed Ayaan Hasan

# ProgChamp – Platform Overview

---

**ProgChamp is a specialized web application designed for publishing and playing community-developed games. It serves as a creative portfolio for student developers at PES University.**

The mission is to foster a healthy ecosystem, where developers can host their projects and receive community feedback through interactive social features, like comments and "Superlikes".

# An Overview on Node.js

---

## Architecture

- JS code behaves differently in different browsers. **solution = node.js**
- Chrome's **v8 engine** embedded in a **C++ program**
- **runtime environment for JS code** outside of a browser
- **I/O intensive** apps
- **non-blocking, event-driven (asynchronous single-thread** model with event queue)
- **open-source** and **cross-platform**

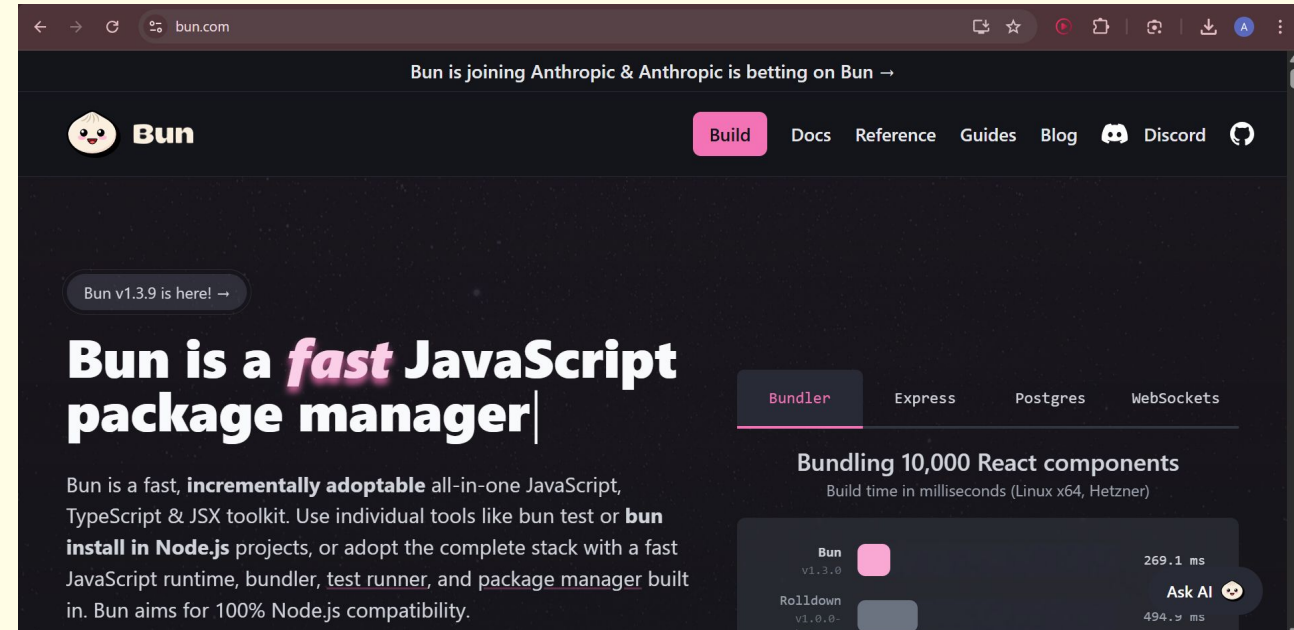
# Installing Bun!

We learnt how to install bun, which is the toolkit we are using to run TypeScript files. We then used it to install the latest versions of all the dependencies...

```
bun install v1.3.9 (cf6cdbbb)

+ @types/bun@1.3.6
+ drizzle-kit@0.31.8
+ @aws-sdk/client-s3@3.969.0
+ @libsql/client@0.17.0
+ drizzle-orm@0.45.1
+ hono@4.11.4
+ zod@4.3.5
+ typescript@5.9.3

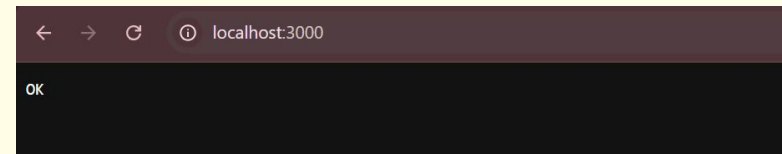
146 packages installed [50.50s]
```



FYI – bun uses JavaScriptCore (JSC) engine, not v8

...and finally ran an example index.ts for the first time!

```
D:\Archita\PES\sem 2\AIEP\ProgChamp\backend\src>bun run index.ts
Backend running on http://localhost:9210
Started development server: http://localhost:3000
```



# Why we're using Bun over npm

---

- bun is written in **Zig** (faster than C++) so it is significantly **faster** than node + npm
- runs **TypeScript** natively
- much **simpler**

Task	npm	Bun
Installing packages	~10-30 seconds	~1-3 seconds
Starting a server	~300ms	~50ms
Running TypeScript	Needs compilation step	Runs directly, no setup

- combination of **runtime** environment (node equivalent) , **package manager** (npm / yarn equivalent), **bundler & test runner** all in one

# Modern Technical Architecture

---



## Frontend

SvelteKit & Svelte with  
TypeScript for  
high-performance reactive  
interfaces and seamless  
transitions.



## Backend

Hono framework running on  
Cloudflare Workers for ultra-fast,  
edge-computed server logic.

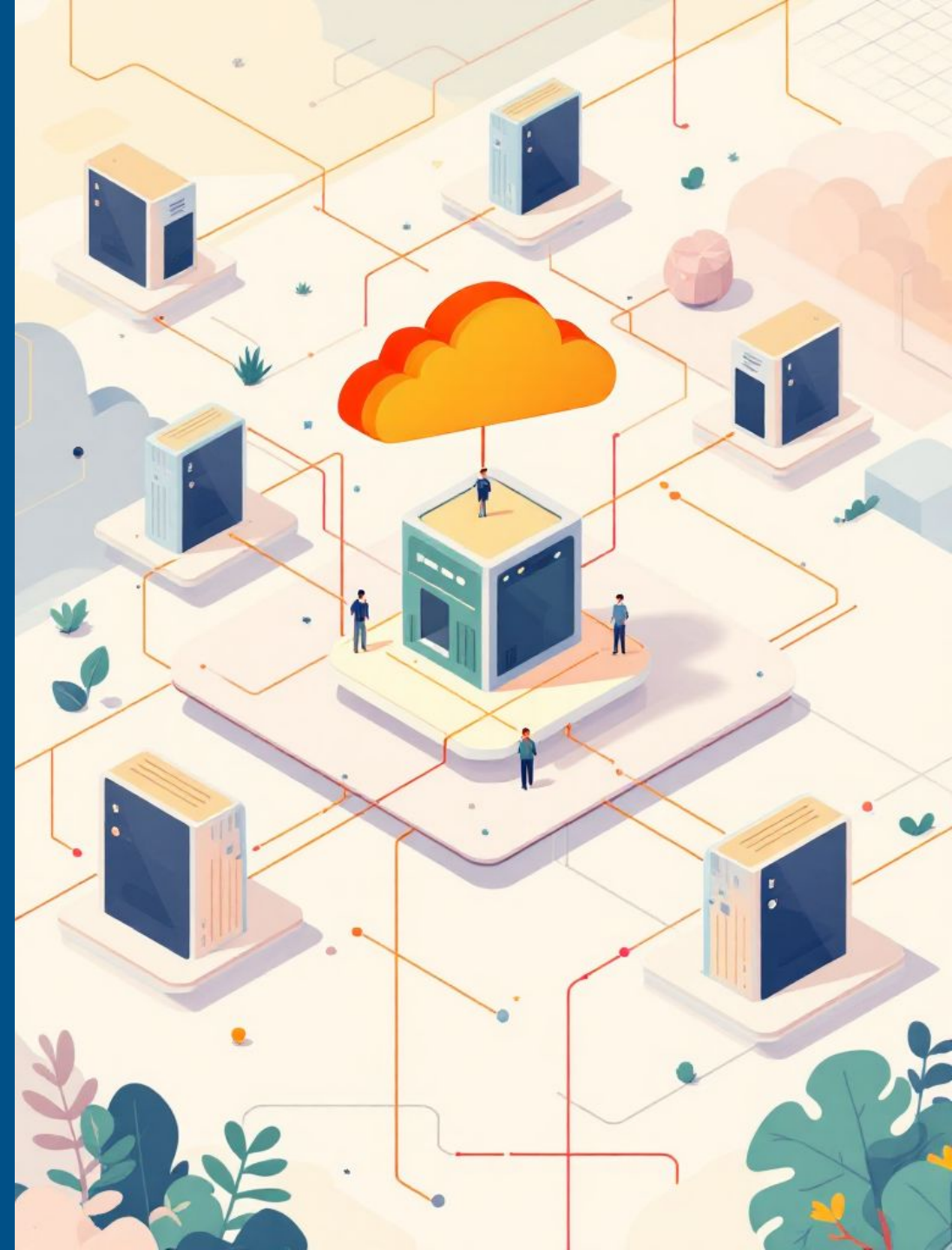


## Storage

Turso (libSQL) with Drizzle ORM  
and Cloudflare R2 for  
decentralized asset  
management.

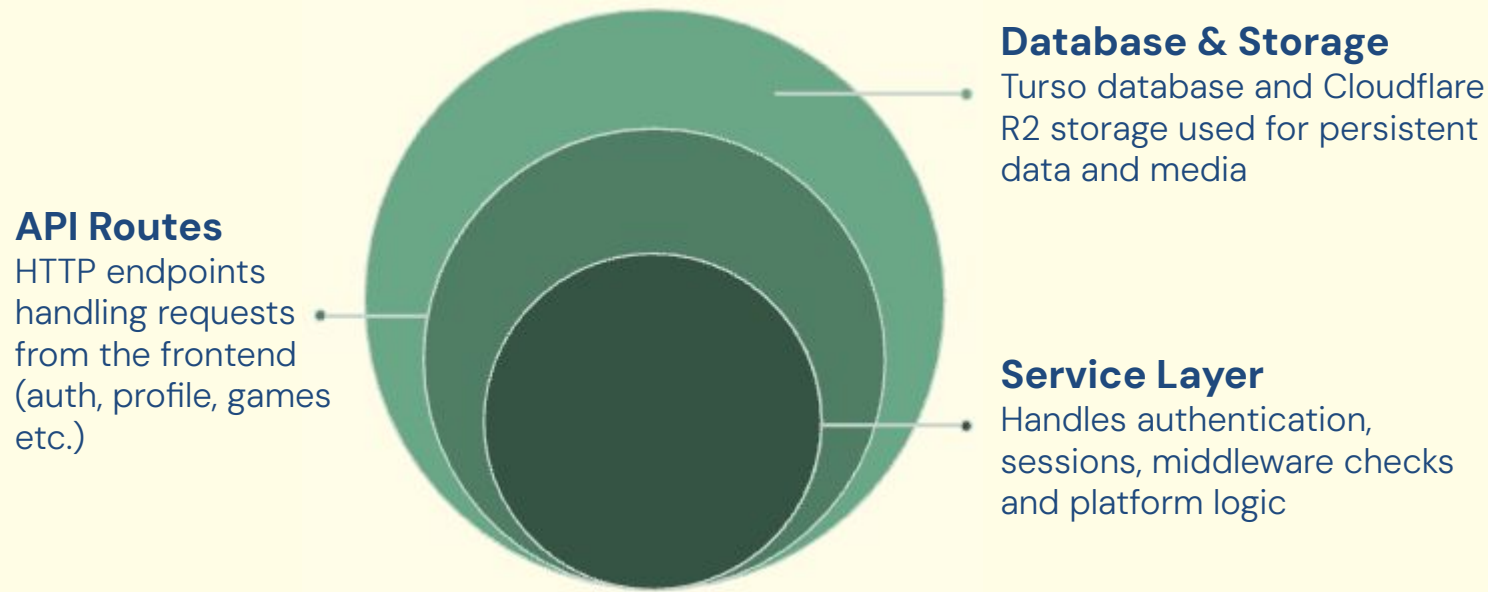
# Backend

The backend acts as the core logic layer of ProgChamp, handling authentication, data management and API operations through a serverless architecture built on Cloudflare Workers



# Backend Structure

---



Conceptual layers of the backend architecture

The modular architecture ensures clean separation of concerns and easy maintenance.

## Key Components

### Hono On Cloudflare Workers

Serverless API built with Hono running on Cloudflare Workers, handling requests close to users.

### Core Responsibilities

Handles authentication, game data, user interactions and media storage/uploads.

### Modular Organisation

- routes/ -> API endpoints (auth, profile, games)
- lib. -> services (OAuth, sessions, middleware, storage)
- db/ -> schema and database logic

### Integration Bridge

Connects to frontend, database (Turso), storage (R2) and hosted games through unified APIs

# Backend Layers & Request Flow

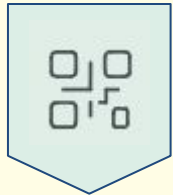
---

How backend layers process incoming requests and coordinate validation, logic and data handling



## **Middleware Layer**

Middleware verifies sessions, user permissions and request validity before requests reach API logic.



## **API Routes**

API routes receive frontend requests and trigger actions such as login, profile updates and game operations.



## **Database Layer (Turso + Drizzle)**

Stores structured platform data such as users, games, reactions, and moderation requests.



## **Storage Layer (Cloudflare R2)**

Stores media files like thumbnails, screenshots, and other game assets separately from the database.

# Authentication & Middleware

## What middleware does in our backend

Middleware runs before API routes and ensures only valid requests reach the application logic.

## Checks enforced by middleware

- **Session validation** → verifies login and loads user identity
- **Account status checks** → blocks deactivated users
- **Role enforcement** → restricts admin-only routes
- **Profile completeness checks** → ensures onboarding is finished

## Why this layer matters

- It centralizes access control across all routes
- Keeps API logic clean and focused on business features
- Provides a consistent security and identity model for the platform



# High Performance Data Layer

---

## What is Turso?

Turso is a distributed database built on libSQL (a fork of SQLite). In ProgChamp, it is used to store structured data such as:

- User Profiles: Storing information about developers and players.
- Game Metadata: Details about the community-developed games, including titles, descriptions, and paths to the hosted files.
- Application State: Managing any persistent data required for the web application to function.

## Why Drizzle ORM?

Drizzle is unique because it offers nearly zero cold-start delay, making it the perfect match for the distributed nature of Cloudflare Workers and Turso DB.

Unlike traditional heavy ORMs, Drizzle translates TypeScript directly to SQL, ensuring the backend stays lean and responsive during high traffic.

# Core Database Architecture

---

## Foundation: Users & Games



**Identity Management:** Users table stores Google IDs, profiles.



**Game Metadata:** Core table tracks URLs, staging IDs, and denormalized counters for optimized reads.



**Session Resilience:** Server-side session storage with cascade deletion for security.



**Performance:** Integer mode timestamps and SQLite-native indexing for Turso speed.

# Relational Logic & Admin Workflow

---



## Asset Mapping

Game Tags (M:N) and Media (R2 storage keys) are stored in decoupled tables with Foreign Keys for easy filtering and asset sorting.



## Interaction Layer

Unified Reaction table (Likes/Dislikes) and separate Superlikes table use Composite Primary Keys to prevent duplicate engagements.



## Staging Workflow

**Game Requests:** Acts as a buffer for admin review. New games remain in "Pending" status until verified, ensuring platform safety.

*\*All relationships are mapped using Drizzle Relations API for seamless type-safe queries.*

## What is Cloudflare R2

- This is where the actual game files, images, and assets live.
- Usually, cloud companies charge you to download your own data. R2 is **free to "get"** data from, which is perfect for a student project.
- It works with standard industry tools but stays within the Cloudflare ecosystem. Making it is **S-3 compatible**.
- This is usually accompanied with **Middleware** that helps keep authorisation in check.

## What is Cloudflare R2

- This is where the actual game files, images, and assets live.
- Usually, cloud companies charge you to download your own data. R2 is **free to "get"** data from, which is perfect for a student project.
- It works with standard industry tools but stays within the Cloudflare ecosystem. Making it is **S-3 compatible**.
- This is usually accompanied with **Middleware** that helps keep authorisation in check.

## THE REQUEST LIFECYCLE

- **Step 1:**A User sends a game file to our URL
- **Step 2:**The Worker catches it it, runs the Auth Middleware to check their validity, and makes sure they aren't a bot.
- **Step 3:**If everything is good,the Worker takes that file and tells the storage:  
`env.BUCKET.put('game-name',file).`
- **Step 4:**R2 saves it instantly, and the Worker sends a success message back to the user.

# Cloudflare Workers

---

## What is Cloudflare Workers?

- serverless platform – automatically manages scaling, pay per use
- runs TypeScript code on Cloudflare's global edge network
- low server latency
- lightweight v8 engine
- works perfectly with Hono (our web framework)

# Frontend



## What is Svelte Js?

- Svelte is a **modern JavaScript framework** for building user interfaces.
- It compiles components into optimized JavaScript at build time.
- It uses a **component-based architecture**.
- It is designed for high performance and simplicity.

## Why Svelte Js?

- Svelte provides excellent performance by **compiling code at build time** instead of using a virtual DOM.
- It reduces boilerplate and simplifies development with **built-in reactivity**.
- It generates **smaller bundle sizes**, leading to **faster load times**.
- It enables rapid development of **modern, scalable** web applications.

Hono is a lightweight, fast web framework for building APIs and backend applications in JavaScript and TypeScript.

- It is **optimized for edge environments** such as Cloudflare Workers, Deno, Bun, and Node.js.
- Hono uses a simple, **minimal routing system** similar to Express but is much smaller and faster.
- It is designed for high performance, making it ideal for **modern serverless** and **edge-based applications**.

## → Hono V/s Express.js

01	Hono works on node, bun, deno and cloudflare workers	Express only works only on node
02	Hono is lightweight and provides faster cold starts	Express follows a traditional server model and is slower
03	Hono provides excellent built in support	Express requires additional typings

# Deliverables

---

## Road Map - What's Next?

### Integration

**Objective:** Connect and finalize full system functionality

- Integrate backend and frontend
- Connect APIs and database
- Ensure real-time data flow
- Test complete user journeys
- Make end-to-end functionality fully operational

### UI/UX Improvements

**Objective:** Enhance user experience and visual appeal

- Improve overall design consistency
- Refine layout and typography
- Enhance mobile responsiveness
- Add animations and smooth transitions
- Optimize loading and interaction feedback

### Administration

**Objective:** Build internal management tools

- Develop Admin Portal
- Create Game Approval Page
- Enable content moderation tools
- Add user management features
- Implement reporting & analytics dashboard



Thank You :D

