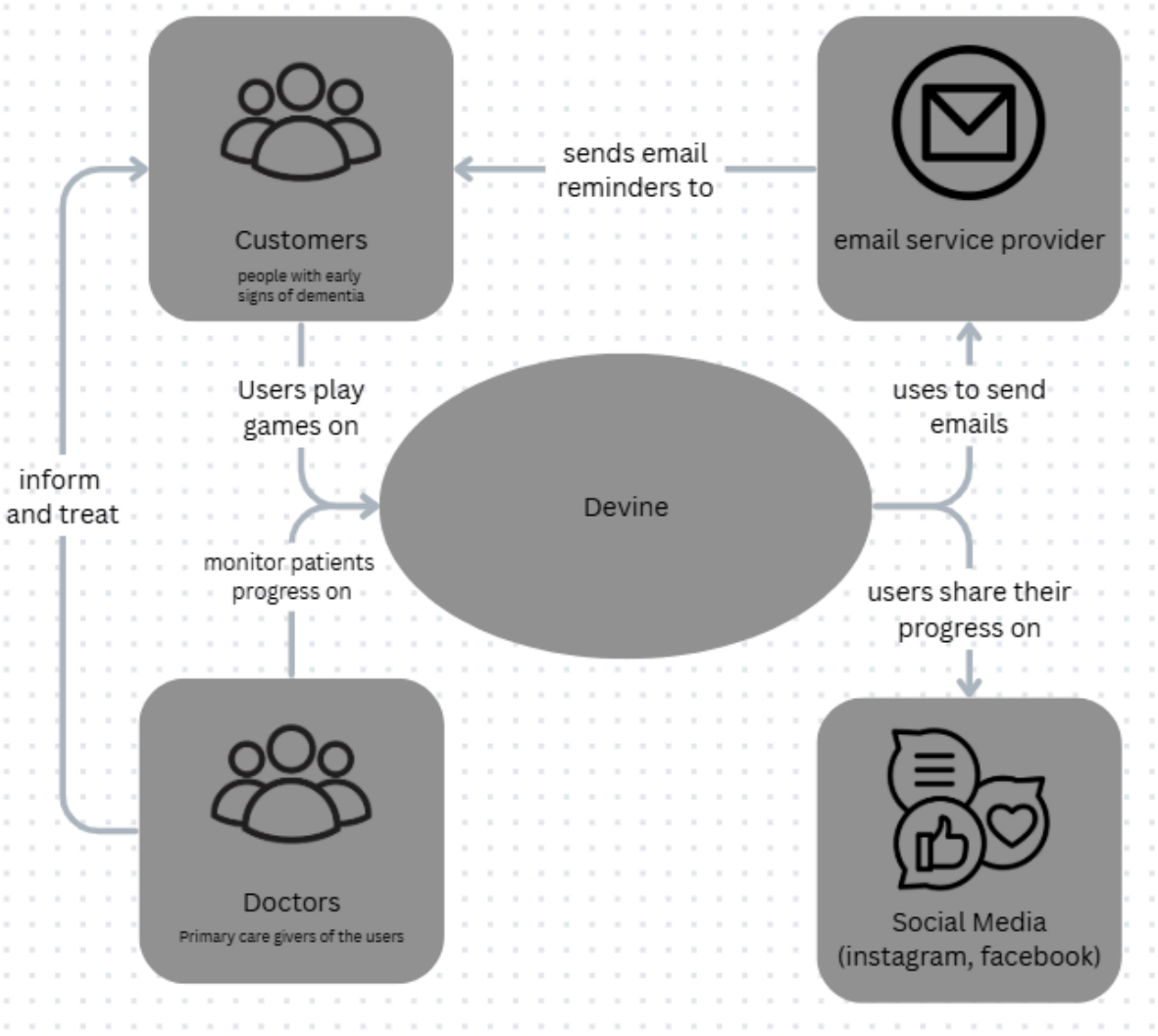
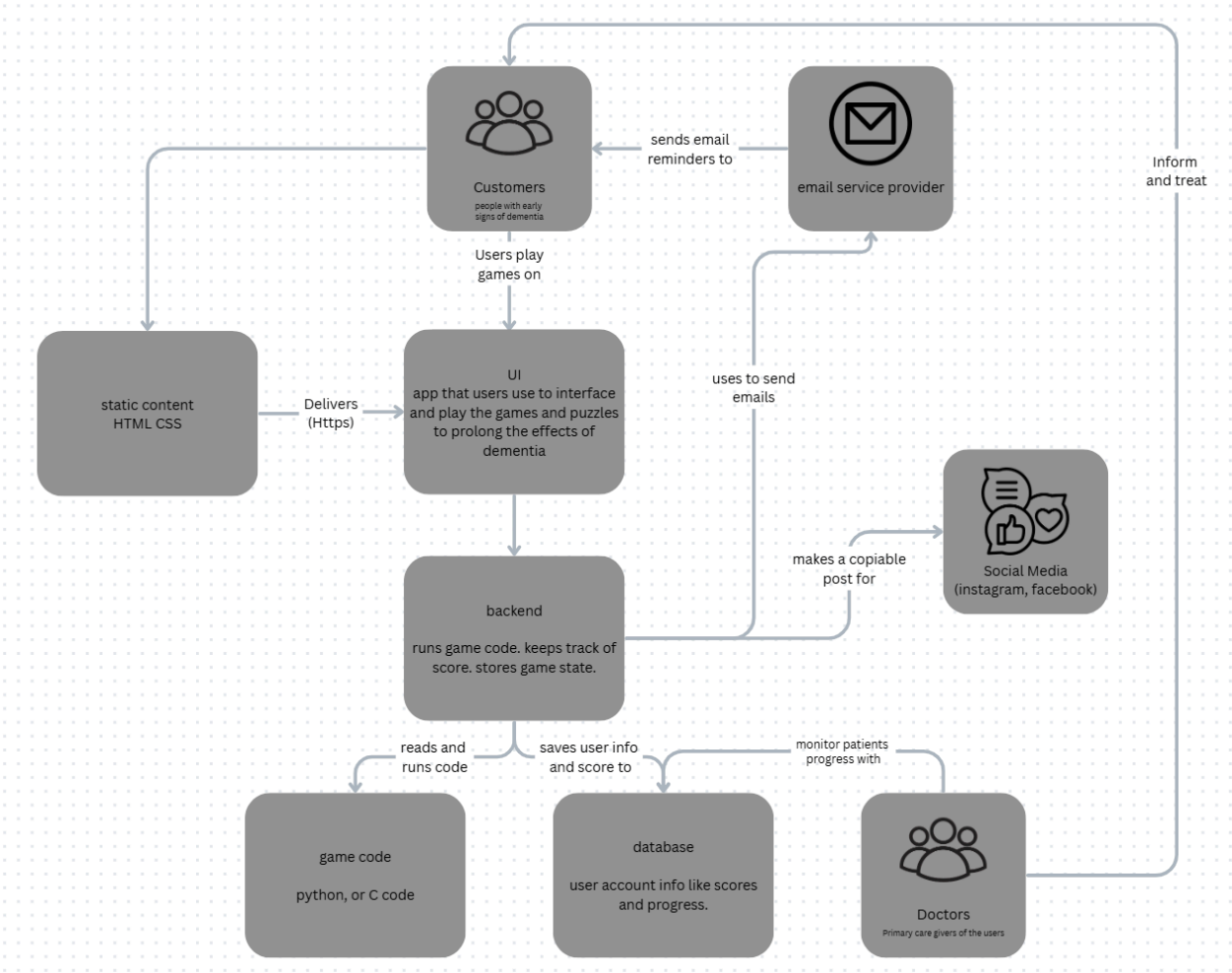
# Devine Design Document

## Diagrams:

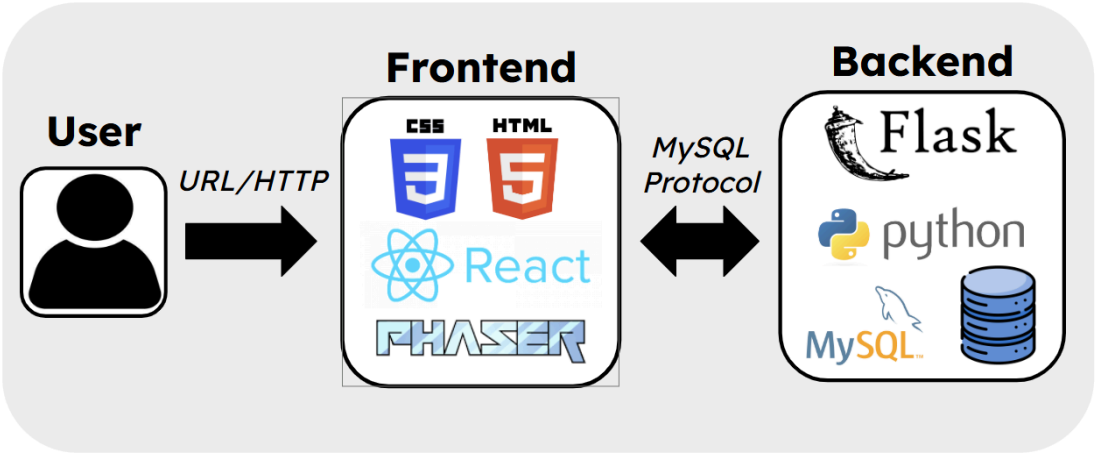
### System Context Diagram:




### Container Diagram:




## Technology Stack:



# Figma:

https://www.figma.com/files/team/1567245597281119459/project/491100214/Senior-Design-Project?fuid=1567260846913524011

---

# Design Trade:

# (Colin McGeary) API & Integration Systems:

- Design 1 Full custom Implementation: A fully custom backend where our team builds all authentication, OAuth integrations, API endpoints, and email connectors manually. This approach offers maximum control over data, security policies, and portability, but comes with heavy development time and long-term maintenance costs.
- Design 2 Backend as a Service: A development option that outsources authentication, hosting, and data management to a BAAS provider, enabling the team to ship an MVP quickly with minimal backend overhead. However, it reduces control over privacy/compliance and increases the risk of vendor lock-in.
- Design 3 A hybrid method: A balanced architecture using a managed authentication provider (Auth0) combined with lightweight microservices behind an API gateway to handle internal logic and integrations. This approach improves security and maintainability while preserving good interoperability and data portability.

**Evaluation Table:**

| | **Design 1** | **Design 2** | **Design 3** |
|---|---|---|---|
| **Security** | **7 (1.4)** | **8 (1.6)** | **9 (1.8)** |
| **Privacy / Compliance** | **9 (1.35)** | **5 (0.75)** | **7 (1.05)** |
| **Development Time (lower is longer)** | **3 (0.45)** | **9 (1.35)** | **7 (1.05)** |
| **Cost (higher is cheaper)** | **9 (0.9)** | **4 (0.4)** | **8 (0.8)** |
| **Scalability** | **6 (0.6)** | **9 (0.9)** | **7 (0.7)** |
| **Maintainability** | **4 (0.4)** | **8 (0.8)** | **8 (0.8)** |
| **Interoperability** | **6 (0.6)** | **6 (0.6)** | **9 (0.9)** |

| | | | |
|---|---|---|---|
| **Data Portability** | **9 (0.9)** | **4 (0.4)** | **8 (0.8)** |
| **Total/10** | **6.6** | **6.8** | **7.9** |

**Reason for Design 3 being Chosen:**

Security and maintainability are critical for Devine's user base; Auth0 provides battle-tested auth features (MFA, breach detection, social login) without requiring the team to implement cryptographic flows from scratch. Development time is reasonable (faster than full custom, slightly slower than pure BaaS), giving a workable MVP within the semester. Scalability & interoperability are excellent: API gateway and microservices scales, and connector platforms/mature APIs make integrations to Mail APIs and social providers straightforward. Cost is moderate but acceptable given the benefits (Auth0/connector platform fees vs. developer cost and risk of in-house security mistakes). Privacy & Data Portability: Hybrid still allows data export from your own DB (microservices own app data), and Auth0 supports user export, so portability is preserved better than pure BaaS. This satisfies our requirements and keeps future compliance work manageable.

---

## (Nathan Galinowski) Database Systems:

- **Design 1: D1-Centralized Relational Database (Cloud SQL Model):** This design uses a single centralized relational database (I.e., Amazon RDS with PostgreSQL) hosted in the cloud. The application's API directly interacts with this database for all data operations, including user profiles, game results, and analytics, and data is stored in normalized tables, with foreign keys linking users, sessions, and performance metrics.
- **Design 2: D2-Distributed NoSQL Database with Streamed Data Pipeline:** This design uses a NoSQL database like MongoDB Atlas or DynamoDB, paired with a real-time data pipeline using tools like AWS Kinesis or Apache Kafka. The pipeline streams game data events (like performance stats) asynchronously to the database, supporting near-real-time analytics.
- **Design 3: D3-Hybrid Relational and Analytics Data Warehouse:** This design combines a transactional relational database (i.e., Amazon RDS) for user and gameplay data with a data warehouse (i.e., Amazon Redshift or BigQuery) for analytics. The data pipeline extracts and transfers summarized data periodically (ETL process) from the main database to the warehouse for trend analysis and reporting.

**Evaluation Table:**

| Criterion | Weight (%) | Design 1: Centralized Relational (Cloud SQL) | Design 2: Distributed NoSQL & Streamed Pipeline | Design 3: Hybrid Relational & Warehouse |
|---|---|---|---|---|
| Data Reliability | 25 | 9 (2.25) | 7 (1.75) | 9 (2.25) |
| Latency (Performance) | 20 | 8 (1.60) | 9 (1.80) | 7 (1.40) |
| Scalability | 15 | 7 (1.05) | 9 (1.35) | 8 (1.20) |
| Security & Compliance | 20 | 9 (1.80) | 8 (1.60) | 9 (1.80) |
| Maintainability | 10 | 8 (0.80) | 6 (0.60) | 7 (0.70) |
| Cost Efficiency | 10 | 9 (0.90) | 7 (0.70) | 6 (0.60) |
| Total Weighted Score | 100 | 9.40 | 7.80 | 8.00 |

Design 1 (Centralized Relational Cloud SQL) was selected because it provides the best overall balance between reliability, security, maintainability, and cost efficiency. For the Devine application, ensuring data accuracy, secure storage of health-related information, and dependable performance outweigh the marginal advantages of a distributed NOSQL system. The centralized relational approach also simplifies development, deployment, and compliance with privacy regulations like HIPAA. While NoSQL or hybrid designs may offer better scalability or analytics capabilities in the long term, the Cloud SQL model meets all current function and non-functional requirements effectively and is easier to maintain during early development and pilot stages.

---

## Joshua Kirby (Game Development Framework):

**Design Options:**

1. *Phaser.js:* A lightweight client-side javascript library used to create 2D games. It is designed to embed simple games on websites while providing an extensive suite of tools to streamline development.
2. *Babylon.js:* A heavyweight online game development javascript library. It is designed for 2D and 3D games with a focus on graphics.
3. *Godot:* An industry standard game engine designed to create 2D games with varying languages. It is known for its extensive toolset to make game development efficient.

**Table:**

| | **Design ID-1** | **Design ID-2** | **Design ID-3** |
|---|---|---|---|
| **Data Speed** | 2.40 (8) | 2.10 (7) | 1.80 (6) |
| **Window Format** | 2.70 (9) | 2.40 (8) | 1.80 (6) |
| **Game Creation Uniformity** | 1.40 (7) | 1.40 (7) | 1.80 (9) |
| **Load Time** | 0.45 (9) | 0.35 (7) | 0.25 (5) |
| **Device Independence** | 1.35 (9) | 1.20 (8) | 0.90 (6) |

**Solution Reasoning:** The Phaser.js library was chosen for Devine due to its lightweight, client side nature. We do not need to incur the lag generated by a graphics intensive library or make 3D games. Nor do we need to figure out the complexities of integrating a normally non-web based game engine to a React application. Phaser.js is an ideal solution as it provides the tools we need to engineer our solution well without adding development complexities.

---

## Daniel Chen (UI/UX System):

### Design ID-1: React + Next.js + Headless A11y (e.g., React Aria / Radix) + Utility CSS

This design uses semantic HTML with accessible primitives, SSR/SSG for performance, Tailwind/utility CSS for tokens, Lighthouse/Axe in CI, and optional PWA. It provides the most direct path to WCAG conformance and Core Web Vitals, with straightforward integration with game canvas. There is excellent web a11y semantics, a mature ecosystem, fast iteration, strong

performance, and easy routing. This choice requires disciplined component design and documentation.

### Design ID-2: Flutter (Web-first, option to package native)

The option utilizes a single codebase. Flutter widgets are rendered to the web with optional iOS/Android packaging later. There is a cohesive widget set and theming across platforms. Consistent UI, rich animations, and good theming are included. However, there are heavier web bundles; the semantic bridge requires diligent a11y work, and there is steeper learning for web audits.

### Design ID-3: React Native (iOS/Android) + React-Native-Web

This option comprises native mobile apps with shared components bridged to web via RN-Web and a Next.js host. There is a strong native mobile UX with shared tokens. A high-quality mobile feel and reuse of logic/tokens provides a good mix of intuitiveness and efficiency. Mapping a11y/semantics to the web adds complexity, however, and there is more integration overhead.

| Criterion | Weight | Design 1 | Design 2 | Design 3 |
|---|---|---|---|---|
| **Accessibility** | 25% | 9 (2.25) | 7 (1.75) | 7 (1.75) |
| **Dementia-friendly usability** | 20% | 9 (1.80) | 7 (1.40) | 7 (1.40) |
| **Performance** | 15% | 9 (1.35) | 7 (1.05) | 8 (1.20) |
| **Responsiveness and input** | 15% | 9 (1.35) | 8 (1.20) | 8 (1.20) |
| **Dev speed and maintainability** | 15% | 8 (1.20) | 7 (1.05) | 6 (0.90) |
| **Offline/PWA** | 5% | 8 (0.40) | 7 (0.35) | 7 (0.35) |
| **Design-system flexibility** | 5% | 9 (0.45) | 8 (0.40) | 7 (0.35) |
| **Total / 10** | 100% | 8.80 | 7.20 | 7.15 |

## Reasons for the Final Choice

We should select Design ID-1 (React + Next.js + headless accessible components). It has the best alignment with accessibility and a dementia-friendly UX. It boasts strong performance, clean integration with game canvas and APIs. There are multiple maintainable tokens/components. All of these combined means it provides the fastest path to a high-quality MVP.