AI SYSTEM ARCHITECTURE DESIGNER

Setting Tools&API Forum

Project1

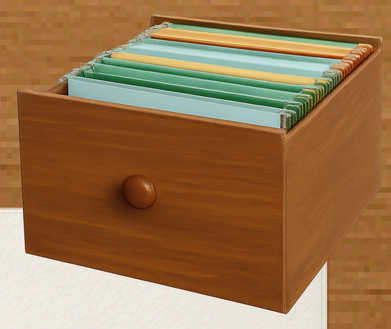
App Type

Core Features

Project Description... |

need inspiration?

👉Generate



Markdown

Export/Print

Solution1

🐍 Snake Game – System Architecture Design Plan

1. System Overview
This is a **web-based snake game** with support for **offline play**, user authentication, and registration.
The system is designed for a **small user base (<100 users)**. It provides basic snake game mechanics and supports **offline-online state synchronization**.

---

2. Architecture Design

2.1 Overall Architecture
The architecture follows a **front-end and back-end separation** model for better scalability and modularity.

- **Front-end**:
  - Framework: **React.js** or **Vue.js**
  - Features:
    - Game canvas and UI rendering
    - User interaction and control
    - Offline caching (via `IndexedDB` or `localStorage`)
    - Service workers for PWA functionality

- **Back-end**:
  - Framework: **Node.js** + **Express**
  - Responsibilities:
    - RESTful APIs for user login, registration
    - Game state saving/loading
    - Synchronization queue support

- **Database**:
  - Options: **MongoDB** (cloud) or **SQLite** (embedded, for local/small-scale)
  - Stores:
    - User credentials
    - Historical game records
    - Offline queue awaiting sync

- **Offline Support**:
  - Game playable without internet
  - Local game sessions can be cached and later synced when online

---

2.2 Technology Stack Summary

Layer	Technology	Role
Front-end	React.js / Vue.js	UI, game canvas, offline cache
Back-end	Node.js + Express	Business logic & APIs
Database	MongoDB / SQLite	User & game data storage
Offline Sync	Service Worker, IndexedDB	Offline gameplay support

---

2.3 Security Considerations

- **Password hashing**: `bcrypt`
- **Authentication**: JWT token-based system
- **Validation**: API input validation & rate-limiting

2.4 Deployment Plan

- **Front-end**:
  - Host on **Netlify**, **Vercel**, or other static site hosts

- **Back-end**:
  - Deploy to **Render**, **Heroku**, or lightweight VPS
  - Use reverse proxy (e.g. Nginx) for route handling

- **Database**:
  - **MongoDB Atlas** (cloud DB)
  - Or **SQLite** (for local minimal setup)

✅ Summary

This lightweight architecture ensures:
- Smooth gameplay experience for a small group of users
- Offline accessibility
- Easy maintenance and deployment