

Draw it or lose it!

# **CS 230 Project Software Design Template**

Version 1.0

## Table of Contents

[**CS 230 Project Software Design Template** 1](#_Toc196499703)

[Table of Contents 2](#_Toc196499704)

[Document Revision History 2](#_Toc196499705)

[Recommendations 3](#_Toc196499706)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | <04/24/25> | Hayde Serrano | Recommendations |

## 

## Recommendations

1. **Operating Platform**:

I recommend using Linux. It’s free to use, very stable, and widely supported. You can run lots of game instances on the same hardware or in the cloud without paying for OS licenses.

1. **Operating Systems Architectures**:

Linux runs everything on one core piece of software the “kernel”, which means fast communication inside the system. We’ll put each game instance in its own container using Docker, so they’re kept separate and easy to start, stop, or move to different servers.

1. **Storage Management**:

* **Images:** Store your 200 game images in object storage like Amazon S3 or a similar service. This keeps them safe, lets you grow your library over time, and makes it fast for players all over the world (using a CDN).
* **Databases & Caches:** Use block storage like AWS EBS or a Kubernetes volume for your game database, PostgreSQL and in-memory cache, Redis. It behaves like a normal disk but is backed up and can survive server failures.

1. **Memory Management**:

On Linux, we set a memory limit on each Docker container so no single game can crash the whole server. Inside each container, the game only keeps a small “window” of upcoming images in RAM for example, five at a time and throws away images it’s already shown. This way, the game runs smoothly without using too much memory.

1. **Distributed Systems and Networks**:

We’ll front everything with a load balancer like NGINX that sends each player to a healthy server. The game servers talk to each other and the database over secure HTTP or WebSockets for real-time updates. If one server goes down, the load balancer sends players to another automatically, so the game stays online.

1. **Security**:

* Encrypt all traffic with TLS so players’ data and game moves stay private.
* Use token-based login OAuth2 with JWTs instead of simple passwords.
* Isolate services with container networking rules so only the parts that need to talk can talk.
* Keep everything backed up and up-to-date with security patches so no one can sneak in through an old vulnerability.