

Draw It or Lose It

# **CS 230 Project Software Design Template-Helen Webster**

Version 1.6

## Table of Contents

[**CS 230 Project Software Design Template-Helen Webster** 1](#_Toc194685222)

[Table of Contents 2](#_Toc194685223)

[Document Revision History 2](#_Toc194685224)

[Executive Summary 3](#_Toc194685225)

[Requirements 3](#_Toc194685226)

[Design Constraints 4](#_Toc194685227)

[Domain Model 5](#_Toc194685228)

[Evaluation 6](#_Toc194685229)

[Recommendations 12](#_Toc194685230)

[References 14](#_Toc194685231)

## [Document Revision History](#_grjogdjh5fi8)

| Version | Date | Author | Comments |
| --- | --- | --- | --- |
| 1.0 | 3/20/2025 | Helen Webster | Provided the Executive Summary and built out requirements/constraint tables. |
| 1.1 | 3/21/2025 | Helen Webster | Analysis of UML Diagram |
| 1.2 | 3/22/2025 | Helen Webster | Comparison of Systems table |
| 1.3 | 3/23/2025 | Helen Webster | Final Recommendation, formatting changes, and addition of references section. |
| 1.4 | 3/23/2025 | Helen Webster | Removed System Architecture section of the template. |
| 1.5 | 4/4/2025 | Helen Webster | Project Two – Evaluation tables added. Updated references section to support Project Two findings. |
| 1.6 | 4/18/2025 | Helen Webster | Project Three – Recommendations section expanded. Updated references section to support Project Three findings. |

## [Executive Summary](#_sbfa50wo7nsh)

The Gaming Room company seeks to expand the reach of its current Android-only game, *Draw It or Lose It*, by developing a web-based version that supports cross-platform access, including OS X, Linux, Windows, and mobile platforms.

The game’s mechanics are inspired by the classic game show *Win, Lose or Draw*, where teams race against time to guess clues based on visual drawings. Unlike the original game show, however, the game will use a vast library of stock images instead of receiving input from users.

The proposed solution involves creating a responsive, browser-based application that replicates the current game’s functionality while introducing features that enable scalability, performance, and user accessibility across devices. This will involve setting up a scalable back-end environment to manage game instances, players, and team interactions, alongside a secure and responsive front-end to support real-time game play.

The Gaming Room staff have indicated they are looking for guidance on the following items:

* Development environment
* Development strategy

Our approach includes delivering a full software design document outlining the requirements, constraints, data flow, and final recommendation of services.

Our goal is to deliver a modular game application that maintains the fun and competitiveness of the original Android app while offering greater accessibility through web-based distribution.

## Requirements

|  |  |
| --- | --- |
| Business Requirements | Details |
| |  | | --- | | **Multi-platform Support** | | Must be accessible on OS X, Linux, Windows, and Android mobile. |
| Web-based Deployment | The new version must be browser-accessible with no installation required. |
| Branding Consistency | Game must maintain identity and gameplay structure consistent with existing app. |

|  |  |
| --- | --- |
| Technical Requirements | Details |
| |  | | --- | |  |   Image Rendering | Must use a large library of stock images rendered progressively (complete by 30s). |
| Game Rounds | Each game consists of 4 rounds, 1 minute each. Other teams have 15s to guess if first team fails. |
| Unique Naming | |  |  | | --- | --- | |  | Game and team names must be unique and checked in real time. | |  | |
| Multi-team Support | Games must support one or more teams with multiple players. |
| Game State Management | |  | | --- | | Only one instance of a game should exist in memory at any time. Must use unique identifiers for game, teams, and players. | |

## [Design Constraints](#_2et92p0)

|  |  |
| --- | --- |
| Constraints | Details |
| |  |  | | --- | --- | | **Architecture** |  | | |  |  | | --- | --- | |  | Application must function in a web-based environment, impacting session management, real-time communication, and server load balancing (ABRAHAM SILBERSCHATZ, 2008). | |
| |  |  | | --- | --- | | **State Synchronization** |  | | |  |  | | --- | --- | |  | Game state must be consistently synchronized across clients in real time, especially with team turns and image rendering progress (ABRAHAM SILBERSCHATZ, 2008). | |
| |  |  | | --- | --- | | **Resource Constraints** |  | | |  |  | | --- | --- | |  | With potential simultaneous users, server memory and storage must be optimized for managing game states and image assets (ABRAHAM SILBERSCHATZ, 2008). | |
| |  |  | | --- | --- | | **Unique Identifiers** |  | | |  |  | | --- | --- | |  | Unique naming and ID generation will require validation to prevent duplicates and maintain data integrity (ABRAHAM SILBERSCHATZ, 2008). | |
| |  |  | | --- | --- | | **Session Handling** |  | | |  |  | | --- | --- | |  | User sessions must persist through disconnections or browser refreshes without losing game progress. | |
| |  |  | | --- | --- | | **Security** |  | | |  |  | | --- | --- | |  | Game data and player interactions must be secured from unauthorized access. | |
| |  |  | | --- | --- | | **Scalability** |  | | |  |  | | --- | --- | |  | Image rendering and gameplay logic must remain responsive across devices and browsers with varying processing power. | |
| |  |  | | --- | --- | | **No Native Drawing Input** |  | | |  |  | | --- | --- | |  | As image input is from a stock library, there will be no support for freehand drawing by players, which affects UI design and game interaction expectations. | |

## [Domain Model](#_8h2ehzxfam4o)

**"The Gaming Room UML diagram. The top of the diagram is labeled as com dot gamingroom. Test boxes are placed in two layers. The first layer has three text boxes and the second layer has four of them. In the first layer, the 'ProgramDriver' textbox points to 'SingletonTester' textbox. The 'ProgramDriver' textbox contains the text 'asterisk main round brackets.' The 'SingletonTester' textbox contains the text 'asterisk testSingleton round brackets.' The arrow between these two text boxes are labeled 'open two angle brackets uses close two angle brackets'. In the second layer, there are 'GameService', 'Game', 'Team', and 'Player' text boxes. The 'GameService' textbox has texts arranged in two layers. The first layer contains games colon List open angle bracket Game close angle bracket, nextGamesId colon long, nextPlayer Id colon long, nextTeamId colon long, and service colon GameService. The second layer contains GameService round brackets, getinstance round brackets colon GameService, addGame open parenthesis name colon String close parenthesis colon Game, getGame open parenthesis id colon long close open parenthesis colon Game, getGame open open parenthesis name colon String close open parenthesis colon Game, getGameCount round brackets colon int, getNextPlayerID round brackets colon long, and getNextTeamId round brackets colon long. The 'GameService' box is connected with the 'Game' textbox with a line labeled 'zero dot dt dot asterisk'.  The 'Game' textbox also contains text in two layers. The first layers contains the text teams colon List open angle bracket Team close angle bracket. The second layer has Game open round bracket id colon long comma name colon String close parenthesis, addTeam open parenthesis name colon String close parenthesis Team, toString round brackets colon String. The 'Game' textbox is connected with the 'Team' textbox with a line labeled 'zero dot dt dot asterisk'. The 'Team' textbox also contains text in two layers. The first layers contains the text players colon List open angle bracket Player close angle bracket. The second layer has Team open parenthesis id colon long comma name colon String close parenthesis, addPlayer open parenthesis name colon String close parenthesis colon Player, and toString round brackets colon String. The 'Team' textbox is connected with the 'Player' textbox with a line labeled 'zero dot dt dot asterisk'. It contains the text Player open parenthesis id colon long comma name colon String close parenthesis and toString round brackets colon String. The 'Game', the 'Team, and the 'Player' boxes point to the 'Entity' textbox in first layer. The 'Entity' textbox contains text in two layers. The first layer has the text id colon long and name colon String. The second layer has Entity round brackets, Entity open parenthesis id colon long comma name colon String close parenthesis, getId round brackets colon long, getName round brackets colon String, toString round brackets colon String.**

**Class Relationships**

* **Inheritance Structure:**
  + Game, Team, and Player all inherit from the base class Entity.
  + Code can be reused and identification becomes standardized across all game components (Lavieri).
* **Singleton Pattern:**
  + GameService is implemented as a singleton, meaning only one instance of the service exists in memory (Lavieri).
  + Supports the business requirement that only one instance of the game may be active at a time.
* **Containment:**
  + A Game contains multiple Team objects.
  + Each Team contains multiple Player objects.
  + This structure supports gameplay involving multiple teams and players.
* **Helper Classes:**
  + ProgramDriver serves as the application’s entry point with its main() method.
  + SingletonTester verifies correct behavior of the singleton implementation.

**OOP Principles Demonstrated**

* **Encapsulation:**
  + Internal data, such as the lists of teams and players, is hidden and accessed through public methods.
  + This protects integrity and enforces proper usage patterns (Lavieri).
* **Inheritance:**
  + Shared behavior is abstracted into the Entity base class.
  + Reduces code duplication and simplifies maintenance (Lavieri).
* **Abstraction:**
  + Entity defines common characteristics such as ID and Name that all game elements share.
  + Allows uniform handling of game objects.
* **Singleton:**
  + Guarantees centralized control of game state and object creation.
  + Prevents conflicting or duplicate instances (Lavieri).

**Requirement Fulfillment (Tech Requirements Table)**

* Supports unique identification and naming for games, teams, and players.
* Enables structured gameplay with one or more teams per game and multiple players per team.
* Only one active game instance exists in memory via the singleton design.
* Provides modularity for future enhancements such as chatting or record scoring.

## [Evaluation](#_2o15spng8stw)

All findings and recommendations in Project Two are directly aligned with and support the requirements identified in Project One. Each operating platform has been evaluated with consideration of its ability to meet the client’s core needs: cross-platform compatibility, responsive web-based deployment, efficient server hosting, and secure, scalable game delivery. The analysis below highlights how each technical decision addresses specific elements from Project One, providing consistency and full support for the development and deployment of *Draw It or Lose It* across web and mobile environments.

|  |
| --- |
| Development Requirements |
| Server side (ABRAHAM SILBERSCHATZ, 2008) |
| Windows |
| Characteristics:   * One-to-one threading model with Win32 API & Thread Pool API * Shared memory via memory-mapped files * Virtual memory via demand paging with clustering * Uses working set min/max and automatic trimming   Advantages:   * Modular system with robust debugging tools * Supports various operating subsystems * Flexible memory management based on workload   Weaknesses:   * Higher resource usage and licensing costs * Manual configuration of shared memory and pipes * Multi-threading introduces complexity in debugging |
| Server side – Project Two |
| Windows |
| Server-based Deployment Support: Yes, Windows supports full web server deployment using IIS (Internet Information Services) or third-party tools like Apache or NGINX. It can host scalable ASP.NET, Java, or Node.js-based applications.  Licensing Costs: Windows Server OS requires a commercial license. Since The Gaming Room is expecting thousands of players, the edition will most likely require a Datacenter with the addition of Client Access Licenses (CALs) for concurrent users. This would make it one of the most expensive options to run. |
| Mac |
| Characteristics:   * POSIX-compliant threading with pthreads * Memory mapping with mmap(), shared memory with shmget() / shmat() * Virtual memory through demand paging and memory compression * Message-based system architecture   Advantages:   * Unix-based, familiar to Linux developers * Efficient memory compression under load * Good for local development and staging environments   Weaknesses:   * Not commonly used for hosting production web servers * Performance impact due to message-passing overhead * Hardware is more expensive and less customizable |
| Server side – Project Two |
| Mac |
| Server-based Deployment Support: It is recommended that macOS host web applications via Apache, Nginx, or Node.js, and is often used for development or staging, but rarely for production. It lacks wide-scale deployment tools and optimized server support.  Licensing Costs: macOS is included with Apple hardware, so there's no additional software license cost—but Apple hardware itself is expensive. Since macOS Server has been discontinued, scalability for large player counts is not practical, making it a less viable choice for hosting for The Gaming Room. |
| Linux |
| Characteristics:   * One-to-one threading model using pthreads * Shared memory via mmap() and POSIX-compliant shmget() / shmat() * No distinction between processes and threads * Uses slab allocator for efficient memory reuse (task\_struct) * Demand paging for virtual memory   Advantages:   * Highly scalable and stable for production servers * Open-source and cost-effective * Strong memory management with slab allocator * Ideal for hosting backend services for mobile/web apps   Weaknesses:   * Requires expertise in Unix/Linux administration * Debugging multithreaded behavior can be complex * Less intuitive for teams unfamiliar with command-line operations |
| Server side – Project Two |
| Linux |
| Server-based Deployment Support: Highly recommended. Linux is the industry standard for web server deployment. Supports Apache, Nginx, Node.js, Tomcat, and containers like Docker with orchestration through Kubernetes. Perfect for hosting scalable, reliable backend services.  Licensing Costs: Linux distributions (like Ubuntu Server, CentOS, or Debian) are free and open source. There are no licensing fees, making it the most cost-effective option for The Gaming Room while still offering enterprise-level performance. |
| Mobile Devices |
| Characteristics:   * Built on Linux kernel with similar memory/threading models * Uses demand paging and pthreads * No disk swapping; uses low memory killer to manage memory pressure   Advantages:   * Optimized for lightweight, concurrent processes * Efficient for mobile apps with predictable usage patterns * Already in use by The Gaming Room   Weaknesses:   * Not suitable for traditional server hosting * Resource-constrained compared to desktop/server platforms * Termination of background apps can affect stability if mismanaged |
| Server side – Project Two |
| Mobile Devices |
| Server-based Deployment Support: Not supported since mobile devices are not designed for hosting web applications. While apps can run local HTTP services for testing or debugging, they lack the resources, stability, and persistence needed for public-facing, multi-user web servers.  Licensing Costs: No relevant licensing applies since this platform isn't suitable for hosting. Mobile devices should remain as clients, not servers, in the system architecture. |
| Client Side (ABRAHAM SILBERSCHATZ, 2008) |
| Windows |
| * Widest end-user reach; must support it * Moderate to high cost due to licensing and hardware diversity * Longest test cycle due to platform variations * Requires thorough QA for legacy support and OS versions |
| Client Side – Project Two |
| Windows |
| Development Process:   * Compatibility with all major browsers (Chrome, Edge, Firefox) on various Windows versions. * Responsive design is key so that the application displays correctly on desktops, laptops, and hybrid devices like a tablet. * Requires comprehensive QA to address a wide range of screen sizes, hardware, and legacy system behaviors, such as:   + Usability Testing   + Accessibility Testing   + Load/Performance Testing   Cost, Time, Expertise:   * Cost: Moderate to high due to variety in hardware and operating system versions. * Time: Longer test cycles due to system variation. * Expertise: Developers must have strong understanding of cross-browser testing and Windows-specific UI quirks. |
| Mac |
| * Higher hardware cost; Apple ecosystem requirements * Medium learning curve; Unix familiarity is helpful * Important for Apple user support if client wants iOS/macOS coverage * Smaller market share, but vital for quality assurance on Apple devices |
| Client Side – Project Two |
| Mac |
| Development Process:   * Must test compatibility with Safari (native browser), along with Chrome and Firefox. * The responsive web app must handle Retina displays and macOS keyboard/mouse interactions (Apple hardware). * Important to include macOS in the QA plan to support iOS browser expectations as well.   Cost, Time, Expertise:   * Cost: High (requires Apple hardware for development and testing). * Time: Moderate due to the more consistent Apple hardware/software environment. * Expertise: Familiarity with Apple ecosystem and UI expectations is helpful; web expertise remains the same. |
| Linux |
| * Low cost due to open-source tooling and OS * Fast setup for developers already familiar with Unix * Rare for end users to run Linux, so less client-facing demand * Ideal for internal tools or server-side development |
| Client Side – Project Two |
| Linux |
| Development Process:   * Must be compatible with Chromium-based browsers and Firefox on common distributions (Ubuntu, Debian). * Focus is less on end users and more on developer/testing environments. * Application should remain functional regardless of desktop environment.   Cost, Time, Expertise:   * Cost: Low—no licensing fees or expensive hardware. * Time: Minimal additional development time; however, not a primary user base. * Expertise: Useful for internal testing/staging environments due to shared architecture with Android servers. |
| Mobile Devices |
| * App already exists for Android—lower initial cost * Expertise already present from existing codebase * Ongoing cost comes from device testing, OS updates, screen resolution adaptation * Fast-growing user base; high ROI for optimization |
| Client Side – Project Two |
| Mobile Devices |
| Development Process:   * Must be mobile-optimized through responsive web design (CSS media queries, touch interface support). * Must be compatible with WebView-based environments, Chrome (Android), and Safari (iOS). * Consider mobile-specific limitations: reduced processing power, battery impact, and variable screen sizes.   Cost, Time, Expertise:   * Cost: Medium—existing Android app reduces initial cost; testing on iOS increases cost due to hardware. * Time: Moderate, due to testing on multiple screen sizes and OS versions. * Expertise: Requires mobile web development skills and device-specific testing across Android and iOS platforms. |
| Dev Tools (Michalowski, 2025) |
| Windows |
| * Languages: Java, C#, JavaScript * IDEs: Visual Studio, IntelliJ IDEA, Eclipse * Build Tools: MSBuild, Maven, Gradle * Testing: NUnit, JUnit, Selenium |
| Dev Tools – Project Two |
| Windows |
| Team Impact:   * Teams may need expertise in .NET/C# for Windows-native apps and JavaScript/Java for web-based cross-platform code. * Integration with Windows-specific APIs (e.g., registry, file system access) may require additional development time. * Good for full-stack teams experienced with Microsoft technologies.   Team Structure:   * Web-focused developers can use JavaScript frameworks; Windows-specific features might require a separate skill set using C#.   Licensing Costs:   * Visual Studio has both free (Community) and paid (Professional/Enterprise) versions. * Most other tools (IntelliJ Community, Eclipse, VS Code, MSBuild) are free for general use. |
| Mac |
| * Languages: Swift (for native), JavaScript, Java * IDEs: Xcode, IntelliJ IDEA, VS Code * Build Tools: Xcode Build, Gradle, Webpack (for web apps) * Testing: XCTest, JUnit, Cypress |
|  |
|  |
| Dev Tools – Project Two |
| Mac |
| Team Impact:   * Requires developers familiar with Swift/Xcode for native Apple features and App Store integration. * Teams also need web developers for building the cross-platform responsive app in JavaScript/HTML/CSS. * Apple hardware is necessary for testing and deployment to iOS/macOS.   Team Structure:   * A separate iOS/macOS-aware team may be needed for testing and debugging Apple-specific UI behavior and Safari browser quirks.   Licensing Costs:   * Xcode is free, but Apple Developer Program is required for App Store distribution and device testing. * Other tools like IntelliJ IDEA Community, VS Code, and Webpack are free. |
| Linux |
| * Languages: Java, Kotlin, Python, C++ * IDEs: Eclipse, IntelliJ IDEA, VS Code * Build Tools: Maven, Gradle * Testing: JUnit, Postman (API), Docker for deployment |
| Dev Tools – Project Two |
| Linux |
| Team Impact:   * Ideal for backend development, CI/CD pipelines, and server deployment. * Developers need comfort with command-line tools, shell scripting, and Unix-like environments. * Testing browser compatibility across desktop Linux environments may be limited but helpful internally.   Team Structure:   * Linux developers can overlap with Android and backend roles, reducing team size through shared skills.   Licensing Costs:   * All listed tools (Eclipse, VS Code, IntelliJ IDEA Community, Docker, Postman) are open source or free, making Linux a low-cost environment. |
| Mobile Devices |
| * Languages: Java, Kotlin (Android) * IDEs: Android Studio, IntelliJ IDEA * Build Tools: Maven, Gradle * Testing: JUnit, Espresso, Firebase Test Lab |
| Dev Tools – Project Two |
| Mobile Devices |
| Team Impact:   * Teams must support both Android and iOS, requiring dual-platform expertise in Java/Kotlin and mobile web development. * Testing on real devices and emulators is necessary due to platform fragmentation (especially Android).   Team Structure:   * Depending on app complexity, separate Android and iOS developers may be needed, or a unified mobile team using cross-platform frameworks (e.g., Flutter, React Native).   Licensing Costs:   * Android Studio and Firebase Test Lab have free tiers; paid tiers depend on usage. * For iOS development, Apple requires a developer license. |

## Recommendations

The following recommendations build directly on the foundational requirements and constraints outlined in Project One, where platform compatibility, performance, scalability, and security were identified for expanding *Draw It or Lose It* beyond Android. Project One established the need for a web-based, multi-platform deployment strategy, as well as server-side reliability and responsive client-side performance. These recommendations apply that groundwork by identifying a suitable server operating platform, architectural details, memory and storage strategies, distributed networking methods, and security practices.

**1. Operating Platform** (ABRAHAM SILBERSCHATZ, 2008)

We recommend using Linux for the server-side of *Draw It or Lose It*. It’s reliable, scalable, and works well with cloud hosting. For the client side, the app should support Windows, macOS, Android, and iOS, using responsive web design and tools like React for cross-platform compatibility.

**Why Linux?**

* Stable and scalable for high-traffic web apps
* Free and well-supported by the developer community
* Compatible with Android, making mobile integration easier
* Easily hosted on cloud platforms like AWS or Google Cloud

**Supporting Rationale:** Supports containerization tools like Docker and Kubernetes for modern deployment, which aligns with the scalable, multi-instance architecture needed for Draw It or Lose It (Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. 2015).

**2. Operating System Architecture** (ABRAHAM SILBERSCHATZ, 2008)

Linux uses a fast and efficient design with strong support for multitasking and real-time processes. It supports:

* Threading through pthreads
* Efficient memory allocation (slab allocator)
* Demand paging for better memory usage
* Easy communication between processes using mmap(), pipes, and shared memory

**Supporting Rationale:** Emphasizes fast inter-process communication (IPC) via shared memory (mmap, shmget) to manage game state between microservices with low latency (ABRAHAM SILBERSCHATZ, 2008).

This makes it perfect for a real-time multiplayer game like *Draw It or Lose It*.

**3. Storage Management**

We recommend using cloud storage (like Amazon S3 or Google Cloud Storage) for images and assets, and a relational database (like PostgreSQL or MySQL) for game data.

* **Cloud Storage:** for drawings, logos, and media files
* **Database:** for user accounts, team info, game sessions, scores

**Supporting Rationale:** Recommends indexed file allocation for fast image retrieval and hybrid serverless architecture (e.g., AWS Lambda) to handle image access and reduce backend load

This setup is flexible, reliable, and scalable for growth (Bashir, 2019).

**4. Memory Management**

Linux handles memory well for high-performance apps:

* Loads only needed memory with demand paging
* Reuses memory efficiently with slab allocation
* Allows threads to share memory for fast performance
* Adapts to memory pressure to avoid crashes

**Supporting Rationale:** Adds detailed memory strategies like direct access and indexed allocation to handle image rendering under load. On Android, memory is also managed well with built-in tools to limit background processes, keeping mobile gameplay smooth (Lenovo, n.d.).

**5. Distributed Systems & Networks** (Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. 2015)

To let the game work across all devices and platforms:

* Use WebSockets for real-time communication between players
* Use REST APIs for account info, game logic, and data syncing
* Add a load balancer to handle heavy traffic
* Use Docker and Kubernetes to manage the game across multiple servers
* Store session data in a shared system like Redis

**Plan for issues like:**

* **Network outages:** Retry failed requests
* **Syncing:** WebSockets keep all devices in sync
* **Session control:** Secure tokens manage player sessions

**Supporting Rationale:** Includes Redis for shared session data, failover strategies (retry logic), and network resilience planning for outages or dropped connections (Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. 2015).

**6. Security** (Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. 2015)

Security is critical and should be built into every part of the system:

* **During communication:** Use HTTPS and secure WebSockets (WSS)
* **Login/auth:** Use OAuth 2.0 and role-based access for admin tools
* **Stored data:** Encrypt sensitive info (like passwords) and media files
* **On mobile and web:**
  + Validate all inputs to prevent attacks
  + Store tokens securely using Keychain (iOS) or Keystore (Android)
  + Monitor activity and keep logs for tracking issues

**Supporting Rationale:** Strengthen with OAuth 2.0 for extensible authentication, encrypt user tokens, and store credentials securely using Keychain (iOS) and Keystore (Android). Add monitoring and logging to detect unauthorized access (Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. 2015).

# References

ABRAHAM SILBERSCHATZ, P. B. (2008). Operating System Concepts. In P. B. ABRAHAM SILBERSCHATZ, *Operating System Concepts* (8th Edition ed., pp. Chapter 3, Chapter 4, Chapter 8, Chapter 9). Wiley.

Bashir, F. (2019, July 15). *What is Serverless Architecture? What are its Pros and Cons?* Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/what-is-serverless-architecture-what-are-its-pros-and-cons/

Craig, W. (n.d.). *10 Important UI Design Considerations for Web Apps*. Retrieved from Web FX: https://www.webfx.com/blog/web-design/10-important-ui-design-considerations-for-web-apps/

Downs, C. (2017, August 19). *How much does it cost to run a game server?* Retrieved from SERVER MANIA: https://www.servermania.com/kb/articles/how-much-does-it-cost-to-run-a-game-server

Drygulski, D. (2025, February 15). *3 Important Things to Consider When Creating a Web App*. Retrieved from netguru: https://www.netguru.com/blog/things-to-consider-when-creating-web-app

Lavieri, D. E. (n.d.). Hands-On Design Patterns With Java;Learn Design Patterns That Enable The Building Of Large-Scale Software Architectures. In D. E. Lavieri. Packt Publishing.

Lenovo. (n.d.). *What is computer memory?* Retrieved from Lenovo: https://www.lenovo.com/us/en/glossary/computer-memory/?orgRef=https%253A%252F%252Fwww.google.com%252F&srsltid=AfmBOorWi2rY3lMXWXGNYwikkq\_E8YsAyTzUdb\_XRZXwXn7NnBdd\_9C6

Michalowski, M. (2025, March 17). *Top 27 Software Development Tools & Platforms [2025 List]*. Retrieved from spacelift: https://spacelift.io/blog/software-development-tools

Rantanen, L. W. (2000, October 1). *Design Considerations for Web-based Applications*. Retrieved from LUKEW IDEATION + DESIGN: https://www.lukew.com/ff/entry.asp?976

Risi, C. (2024, July 5). *The importance of having detailed technical requirements in software development*. Retrieved from CRAIG RISI: https://www.craigrisi.com/post/the-importance-of-having-detailed-technical-requirements-in-software-development

Spolsky, J. (2000, September 27). *WhatTimeIsIt.com*. Retrieved from Joel on Software: https://www.joelonsoftware.com/whattimeisit/

Villamizar, M., Garcés, O., Castro, H., Verano, M., Salamanca, L., Casallas, R., & Gil, S. (2015). *Evaluating the impact of microservice architectures using containers*. Journal of Systems and Software, 116, 94–106. https://doi.org/10.1016/j.jss.2016.01.017

Zhang, A. (2018, July 13). *How to write a good software design doc*. Retrieved from freeCodeCamp: https://www.freecodecamp.org/news/how-to-write-a-good-software-design-document-66fcf019569c/