10/7/24

PART ONE OF MY PROJECT *NOT DOING RIGHT NOW*:
Running an accessible workshop program that has the ability to teach artists in their fields (photography, painting, sculpting, videography, etc)
PART TWO OF MY PROJECT *WHAT I'M DOING RIGHT NOW*:
After working hard at the end all the students will have a virtual showcase of their work, an exhibition. They will be able to invite their family and friends.
WHAT I LEARNED TODAY:
To create a safe game where there cannot be any uninvited members similar to a body guard at an event I need a private hosted server. I highlighted all the information which is important and what I learned. I ended up going with Unity as my game engine!

1. Use Standard Technologies: Choose common protocols (HTTP, WebSockets) and databases (MySQL, MongoDB) to ensure compatibility across hosting platforms.
2. Containerization & Automation: Use tools like Docker for containerization and Kubernetes for deployment, so your server can be easily moved and replicated.
3. Separate Components: Break down your server into distinct modules (e.g., authentication, data storage) so each part can be independently updated, replaced, or scaled.

how to design for flexibility?

- Design for Flexibility: Use modular server code and standard protocols to make switching hosting easier.
- Prepare & Test: Set up and test the new hosting environment before fully moving, ensuring all data and configurations work correctly.
- Smooth Transition: Transfer data, redirect traffic, and monitor closely to catch any issues quickly after the switch.

Is it easy to switch a game from one way of hosting to another form of hosting ex) self-hosting to cloud hosting?

Unity
Pros:

1. Ease of Use & Learning Curve:
   - Beginner-Friendly: Easier to learn for new developers, with a simpler interface.
   - Large Asset Store: A vast marketplace with ready-to-use assets, tools, and scripts, speeding up development.
2. Cross-Platform Support:
   - Unity supports over 25 platforms, including mobile (iOS/Android), web (WebGL), desktop (Windows/Mac/Linux), and consoles.
3. 2D & 3D Game Development:
   - Highly flexible for both 2D and 3D projects, making it suitable for a wide variety of game styles.
4. Large Community & Documentation:
   - Extensive community support, tutorials, forums, and official documentation make it easier to find help and resources.
5. Performance:

- Highly optimized for mobile development and runs smoothly on a wide range of devices, including older hardware.

Cons:

1. Graphics & Visual Quality:
   - While capable of high-quality graphics, Unity’s default rendering options are often considered less sophisticated compared to Unreal’s out-of-the-box graphics.
2. Script-Based Development:
   - C# scripting is the main method for development. While powerful, it may be more challenging for those who prefer visual scripting.
3. Licensing Fees:
   - While there’s a free version, more advanced features (Unity Pro) require a subscription fee, especially if your revenue exceeds a certain threshold.

Unreal Engine

Pros:

1. High-Quality Graphics & Visuals:
   - Unreal is known for its high-fidelity visuals and powerful rendering capabilities, providing a cinematic look with advanced shaders and lighting (good for AAA and realistic visuals).
2. Blueprint System (Visual Scripting):
   - Allows for visual scripting through its Blueprint system, which is a major advantage for non-programmers and those who want rapid prototyping without coding.
3. Built-in Tools:
   - Unreal Engine comes with built-in tools for creating a variety of game features, like AI, physics, and networking, reducing the need for third-party plugins.
4. Free Until Revenue Threshold:
   - Completely free until your game earns more than $1 million in revenue, after which a royalty fee (5%) is applied.

Cons:

1. Steeper Learning Curve:
   - More complex to learn, with a steeper curve for beginners. The engine’s interface and toolset can be overwhelming if you’re not familiar with game development.
2. Performance Requirements:
   - Unreal Engine games, especially those with high-fidelity graphics, are more demanding on hardware, which may impact performance on lower-end devices, particularly mobile.
3. Less Suited for 2D Games:
   - Unreal is mainly focused on 3D development and is less optimized for 2D projects compared to Unity.
4. Build Size:
   - Games built with Unreal Engine can be larger in file size, which could be a concern for mobile or web-based projects.

When to Choose Each:

- Choose Unity if: You’re creating a 2D game, need cross-platform flexibility (especially mobile), want something easier to learn, or have lower hardware requirements.

- Choose Unreal if: Your game requires top-tier graphics, realistic visuals, more sophisticated built-in tools, or you prefer visual scripting through Blueprints.

Ultimately, both engines are highly capable, but the best choice depends on the specific needs and goals of your virtual exhibition project. For a visually stunning experience with less coding, Unreal might be ideal. For a more flexible, lightweight approach, Unity is often better.

Which is better: Unity or Unreal Engine?

“Private server hosting” means creating a dedicated server that is only accessible to a select group of users, rather than being open to the general public. For your virtual exhibition game, this would involve setting up a server where only your students and their invited guests can connect and access the exhibition.

To implement private server hosting, here’s how you could go about it:

1. **Choose a Game Engine/Platform:** Use a game engine like Unity or Unreal Engine to build your game, which both support networking features for multiplayer experiences.

Which is better: Unity or Unreal Engine?

2. **Set Up the Server:** You can host your server in several ways:
   - **Self-Hosting:** Run the server from a physical machine that you control. This option requires maintaining your hardware and ensuring it’s always online when needed.
   - **Cloud Hosting Services:** Use cloud providers like AWS, Google Cloud, or Microsoft Azure to host your server. These platforms offer scalable hosting solutions and security options.
   - **Dedicated Game Server Providers:** For game-specific hosting, services like PlayFab, Photon, or even services specific to engines like Unity’s “Multiplayer” tools can be used.

WebStockets? Is it easy to switch a game from one way of hosting to another form of hosting ex) self-hosting to cloud hosting?

3. **Access Controls:**
   - **Authentication System:** Implement an authentication system (username/password, email login, or token-based access) to ensure only approved users can join.
   - **Invitation Codes or Whitelists:** Generate codes or lists of approved usernames that are allowed access to the server. You can manually approve participants or automate the process through an invitation system.
4. **Server Security and Privacy:** Make sure the server has proper security measures to prevent unauthorized access. This includes firewalls, encryption for communication, and regular updates to maintain security.
5. **Testing Access Controls:** Before launching, test the server thoroughly to ensure that only approved users can connect and access the exhibition.

This approach allows you to maintain control over who accesses the virtual exhibition, ensuring it remains a private space for your students and their invited guests.

what does "private server hosting" mean? and how can I implement that?

Creating a game as a platform for the virtual exhibition can be a safe option for limiting access. You can achieve this by implementing access controls, such as login credentials, invitation codes, or private server hosting. This allows only students and their guests to join the exhibition, ensuring privacy and exclusivity.

Additionally, using a game environment can enhance engagement, providing an interactive experience while controlling who can access and participate. It's also important to consider using a secure platform or server to protect user data and maintain control over access permissions effectively.

i have an idea to make a virtual exhibition where people can tune in with a game. i want this to be the second part of an entire projecti have in mind wehre i provide virtual FREE workshops to teach people about art i want to make this the most accesabile as possible however i have the following questions:

1- would creating a game and publishing it be the safest option since i only want this exhibition to be accesible by the students and their invited guests?

10/2/24

Week 1: Planning and Basic Learning

- **Day 1-2: Define the Project Scope**
  - Identify the features of the website/app (e.g., user registration, hosting workshops, galleries for exhibitions).
  - Choose whether it will be a web app or mobile app.
- **Day 3-4: Learn the Basics of Python and Web Development**
  - Learn Python fundamentals (data types, control structures, functions).
  - Study web technologies: HTML, CSS, and JavaScript basics.
- **Day 5-7: Learn Frameworks & Tools**
  - Explore Python frameworks for web development: Django or Flask.
  - Decide on tools for front-end development (e.g., React for JavaScript, Bootstrap for CSS).

Week 2: Development Setup and Backend Basics

- **Day 8-9: Set Up Development Environment**
  - Install necessary tools: Python, Django/Flask, a code editor (e.g., VS Code).
  - Set up Git for version control.

- **Day 10-11: Backend Development (Part 1)**
  - Create a simple Django/Flask project.
  - Learn how to set up URLs, views, and templates.
- **Day 12-14: Database Integration**
  - Learn how to set up and interact with databases (e.g., SQLite).
  - Practice creating models for user registration and workshops.

Week 3: Frontend Development and User Interface

- **Day 15-17: Designing the Frontend**
  - Work on HTML structure for your website/app.
  - Style with CSS and use JavaScript for interactivity.
- **Day 18-19: Integrating Frontend with Backend**
  - Learn how to pass data between front-end forms and back-end views.
  - Implement user registration and login functionality.
- **Day 20-21: Building Workshop and Exhibition Features**
  - Create pages for workshops and exhibitions.
  - Implement a gallery or content upload feature.

Week 4: Advanced Features and Testing

- **Day 22-24: Advanced Python and Web Development Concepts**
  - Learn more about user authentication, file handling, and API integration.
  - Implement user roles (e.g., host, attendee).
- **Day 25-26: Testing and Debugging**
  - Test all parts of the app, ensuring functionality for all features.
  - Debug any issues found during testing.
- **Day 27-28: Add Polish to UI/UX**
  - Enhance the user interface for better experience.
  - Make sure the website/app is responsive (works on mobile).

Week 5: Final Touches and Launch

- **Day 29-30: Final Testing and Review**
  - Perform a full test of all features and functionalities.

    - Get feedback from peers or users.
- **Day 31-32: Prepare for Launch**
    - Deploy the app on a hosting service (e.g., Heroku, AWS).
    - Finalize any documentation (e.g., user guide, code comments).

Week 6: Workshop and Exhibition Preparation

- **Day 33-34: Workshop Setup**
    - Plan how workshops will be hosted (live sessions, recorded content).
- **Day 35-36: Exhibition Setup**
    - Organize exhibition content (images, videos, documents).
- **Day 37-38: Marketing & Launch Event**
    - Spread the word about your platform.
    - Set a date for the workshop/exhibition launch.

9/30/24

Prompt engineering

Interests:

- Photography
- Journalism
- Politics
- Reading

Ideas:

- Photography exhibition
- Photography class

LIFE OF A HIGH SCHOOL STUDENT - GAME

- Real life photos
- 3d animation
- Coding game?

Concept:

GONE HOME?

Real-life Photos: You can use real-life photos as backgrounds, environments, and even character models (with permission, of course!). This gives a unique, authentic feel.

3D Animation: You could use 3D animation for the characters, objects, and interactions. This allows for dynamic movement and more engaging gameplay.

Coding Game? This is where things get interesting! You could create a game that involves coding elements. Think of it like this:

Option 1: Coding Challenges: The player could face coding puzzles or mini-games to solve problems in the game, like fixing a computer glitch or building a website for the school club.
Option 2: Visual Programming: You could use a visual programming language (like Scratch) that makes coding more accessible for beginners. Players could create their own mini-games or animations within the game world.
Option 3: Code-Based Choices: The game could present the player with choices, and the player's coding knowledge could influence the outcomes. For example, a character might need to write a program to win a coding competition, and the player's code determines their success.
Gameplay Ideas:

Daily Routine: The player could experience a typical high school day, attending classes, participating in extracurricular activities, socializing with friends, and making choices that impact their grades, relationships, and overall experience.
Career Exploration: The game could incorporate elements of career exploration, allowing players to experiment with different subjects and activities to discover their passions and interests.
Social Interaction: Players could build relationships with other characters, participate in social events, and navigate the complexities of high school friendships.
Decision-Making: The game could present players with challenging decisions, requiring them to weigh the consequences of their actions and make choices that align with their values.
Technical Considerations:

Game Engine: You'll need a game engine to develop your game. Popular options include Unity, Unreal Engine, and Godot.
Programming Language: The choice of programming language will depend on the game engine you select.
Asset Creation: You'll need to create or acquire assets like 3D models, textures, sound effects, and music.
Creative Considerations:

Storytelling: Consider the overall narrative arc of the game. Will it be a linear story or a more open-ended experience?
Character Development: Create engaging and relatable characters that players can connect with.
World-Building: Design a believable and immersive high school environment that feels authentic.
Remember:

Target Audience: Who is your game for? Think about the age range and skill level of your players.
Scope: Start with a manageable scope. You can always add more features and content later.
Collaboration: If you're not familiar with coding or game development, consider collaborating with others who have expertise in these areas.