Unity Game Builder Project

Custom 3D Model Upload System

Generated on: November 06, 2025

1. Project Overview

What We Want to Achieve

Build a web-based system that allows users to upload custom 3D character models via a website, automatically replace placeholder models in a Unity project on the server, trigger automated builds to generate game executables, and download the customized game to play on their machines.

Business Value

- Customization: Users can personalize games with their own 3D models
- Automation: No manual Unity work required for each custom build
- Scalability: Multiple users can create custom builds simultaneously
- User Experience: Simple web interface accessible to non-technical users

2. System Architecture

The system consists of three main components: a web frontend for user interaction, a backend API server for processing, and the Unity project with build automation.

User's Browser → Web Interface (HTML/CSS/JavaScript)
↓ HTTP Requests
Backend API Server (Node.js) → File Upload, Build Trigger, Status Tracking
↓ File Operations & Command Execution
Unity Project → Model Replacement → Unity Editor CLI → Build Output (.exe)

3. What Needs to Be Done

My Responsibilities (Backend/Web Developer)

A. Backend API Development

- Create Node.js/Express server
- Implement file upload handling (multer)
- Implement model replacement logic
- Implement Unity build trigger (command-line execution)
- Implement build status tracking system
- Implement file download serving
- Error handling and logging

B. Web Frontend Development

- Create upload interface (drag & drop + file picker)
- Create build trigger UI
- Create download interface
- Implement status/progress display
- Implement polling for build status
- User feedback and error messages

C. File Management

- Organize uploaded model storage
- Manage Unity project file paths
- Handle build output storage
- Implement file cleanup (optional)

D. Integration & Testing

- Coordinate with Unity developer for configuration
- Test end-to-end workflow
- Handle edge cases and errors
- Performance optimization

Unity Developer's Responsibilities

- Create Unity build script (BuildScript.cs)
- Configure Unity project structure
- Set up model import settings
- Configure build settings
- Provide configuration details (paths, method names, etc.)
- Test Unity build process independently

4. How to Implement

Phase 1: Setup & Configuration (1-2 days)

- Get information from Unity developer (paths, configurations)
- Initialize Node.js project and install dependencies
- Create project structure (server, frontend, config files)

Phase 2: Backend Development (3-4 days)

- Create server with file upload endpoint
- Implement model replacement logic
- Implement build trigger with Unity CLI
- Implement status tracking system
- Implement download service

Phase 3: Frontend Development (2-3 days)

- Create upload interface with drag & drop
- Create build trigger UI
- Create download interface
- Implement status polling for real-time updates

Phase 4: Integration & Testing (2-3 days)

- Integration testing with Unity developer
- Error handling and edge cases
- Security and optimization
- Final testing and bug fixes

5. Technology Stack

Backend

- Node.js - Runtime environment
- Express.js - Web framework
- Multer - File upload handling
- fs-extra - Enhanced file system operations
- child_process - Execute Unity build commands

Frontend

- HTML5 - Structure
- CSS3 - Styling
- JavaScript (Vanilla) - Interactivity
- Fetch API - HTTP requests

Infrastructure

- File System - Store uploaded models and builds
- Unity Editor CLI - Automated builds
- Web Server - Serve API and frontend

6. Key Features

User Features

- Drag & drop 3D model upload
- Real-time upload progress
- One-click build trigger
- Build status monitoring
- Direct executable download
- Clear error messages

System Features

- Automatic model replacement
- Automated Unity builds
- Build status tracking
- File validation
- Error handling
- Logging

7. Workflow Example

1. User opens website
2. User uploads 3D model (.fbx file) → Backend validates and stores
3. Backend replaces model in Unity project → Copies to Assets/Models/Character.fbx
4. User clicks 'Build Game' → Backend triggers Unity build command
5. Frontend polls build status → Shows 'Building...' status every 2 seconds
6. Build completes → Backend detects .exe file → Updates status to 'completed'
7. User downloads game → Backend streams .exe file → User saves and plays

8. Dependencies & Coordination

Required from Unity Developer

- Unity project access (read/write permissions)
- Unity Editor installation path
- Build script implementation
- Configuration details (paths, method names)
- Testing support

Required Infrastructure

- Server with Node.js installed
- File system access to Unity project
- Unity Editor installed on server
- Sufficient disk space for builds
- Network access for web interface

9. Timeline Estimate

Phase	Task	Duration	Dependencies
1	Setup & Configuration	1-2 days	Unity dev provides config
2	Backend Development	3-4 days	Phase 1 complete
3	Frontend Development	2-3 days	Backend API ready
4	Integration & Testing	2-3 days	All components ready
Total		8-12 days	

10. Risks & Considerations

Technical Risks

- Unity build failures - Need proper error handling
- File system permissions - Ensure proper access rights
- Large file uploads - Implement size limits and validation
- Build time variability - Implement proper status polling
- Concurrent builds - May need queue system for multiple users

Security Considerations

- File upload validation - Prevent malicious files
- Path traversal protection - Validate file paths
- Rate limiting - Prevent abuse
- Authentication (future) - Control who can build
- File cleanup - Manage disk space

Scalability Considerations

- Multiple concurrent builds - May need job queue (Bull/Redis)
- Storage management - Cleanup old builds
- Database for build tracking (optional - currently in-memory)
- Load balancing (if multiple servers)

11. Future Enhancements (Optional)

- User authentication system
- Build history and management
- Multiple model uploads (characters, props, etc.)
- Build customization options (game settings, levels)
- Email notifications when build completes
- Build queue system for multiple users
- Analytics and usage tracking
- Docker containerization for easier deployment

12. Success Criteria

Functional Requirements

✓ Users can upload 3D models successfully
✓ Models are correctly replaced in Unity project
✓ Build process completes successfully
✓ Users can download working executables
✓ System handles errors gracefully

Non-Functional Requirements

✓ Upload completes in reasonable time (< 1 min for typical files)
✓ Build status updates in real-time
✓ System is stable and doesn't crash
✓ Error messages are user-friendly
✓ Code is maintainable and documented

13. Deliverables

Backend API Server

- Complete Node.js/Express server
- All API endpoints implemented
- Error handling and logging

Web Frontend

- Complete HTML/CSS/JavaScript interface
- Responsive design
- User-friendly UI/UX

Documentation

- API documentation
- Setup instructions
- Configuration guide
- Deployment guide

Testing

- Integration tests
- End-to-end workflow verification
- Error scenario testing

14. Next Steps

Immediate Actions:

- Get configuration details from Unity developer
- Set up development environment
- Create project structure

Week 1:

- Complete backend API development
- Basic frontend implementation
- Initial testing

Week 2:

- Complete frontend development
- Integration with Unity developer
- End-to-end testing
- Bug fixes and refinements

Summary

This project creates a web-based system for customizing Unity games with user-uploaded 3D models. The backend handles file uploads, model replacement, and automated builds. The frontend provides a simple interface for users. Implementation takes approximately 8-12 days, with coordination needed from the Unity developer for configuration and testing.

The system automates the entire customization and build process, making it accessible to non-technical users while maintaining the flexibility of Unity game development.