Bilkent University
Department of Computer Engineering

Senior Design Project
Team ID: T2521
Project Name: Ubien: The Immersive Inquiry Based Learning Platform

Final Report

Group Members:

22203367, Ahmet Deniz Gelir, deniz.gelir@ug.bilkent.edu.tr
22202254, Efe Can Tatar, can.tatar@ug.bilkent.edu.tr
22201820, Kemal Onur Özkan, onurozkan@ug.bilkent.edu.tr
22001736, Eren Uslu, eren.uslu@ug.bilkent.edu.tr
22203328,Simay Uygur, simay.uygur@ug.bilkent.edu.tr

Supervisor:

Uğur Güdükbay

Course Instructors:

İlker Burak Kurt

Mert Bıçakçı

01.05.2026

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfilment of the requirements of the Senior Design Project course CS491/2.

Contents

1. Introduction **4**
2. Requirements Details **4**
2.1 Functional Requirements 4
2.1.1 System Functionality 4
2.1.2 User Functionality 5
2.2 Non-functional Requirements 6
2.2.1. Usability 6
2.2.2. Reliability 6
2.2.3. Performance 6
2.2.4. Supportability 6
2.2.5. Scalability 7
3. Final Architecture and Design Details **7**
3.1 Overview 7
3.2 Subsystem Decomposition 7
3.3 Backend Layer 8
3.4 Client Layer 8
3.5 Database Layer (Persistence) 9
4. Development/Implementation Details **10**
4.1 Client Layer 10
4.1.1 Dashboard UI 10
4.1.2 Chat Interface 11
4.1.3 Course Viewer 11
4.1.4 Unity WebGL Runtime 12
4.2 Backend Layer 13
4.2.1 Functionality Managers 13
4.2.2 LLM Communication Handler 14
4.2.3 Animation Controller 14
4.2.4 PDF Processing Pipeline 14
4.3 Cloud Layer 15
5. Test Cases and Results **16**
5.1 Functional Test Cases 16
5.2 Non-Functional Test Cases 29
6. Maintenance Plan and Details **34**
6.1 Limitations of the Server and External API Dependencies 35
6.2 Version Checking and Source Control 35
6.3 Change Requests and Issue Tracking 35
6.4 Maintaining the PDF and LLM Pipelines 36
6.5 3D Avatar and Animation Maintenance 36
7. Other Project Elements **36**
7.1. Consideration of Various Factors in Engineering Design 36
7.1.1 Constraints 36
7.1.2 Standards 37
7.2. Ethics and Professional Responsibilities 37
7.3. Teamwork Details 38
7.3.1. Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives 38

7.3.2. Helping creating a collaborative and inclusive environment..... 39
7.3.3. Taking lead role and sharing leadership on the team..... 39
7.3.4. Meeting objectives..... 39
7.4 New Knowledge Acquired and Applied..... 39
8. Conclusion and Future Work.....40
9. Glossary.....41
10. References.....42

1. Introduction

It is commonplace in the present day to hear students bemoan that while they have "learned" a great deal on paper, they have retained only a fraction of the information they gathered over the course of their education. It is natural, then, to ask: what has been the point of learning all that information if it functioned merely as a temporary exercise in memorization? The current paradigms of education, from structured lectures to solution-driven problem sets and exam-oriented milestones, are built around the efficient delivery and assessment of knowledge rather than the cultivation of genuine understanding. In such systems, students often become passive recipients, carrying information only long enough to reproduce it when asked, and quickly losing it afterward.

This widespread pattern reveals a fundamental problem: the absence of active self-discovery. Educational psychology and centuries of pedagogical tradition alike affirm that concepts uncovered through one's own reasoning are remembered more deeply, understood more clearly, and integrated more permanently into one's intellectual framework [1]. Yet most modern educational tools unintentionally short-circuit this process. They present polished explanations, prepackaged insights, and step-by-step solutions that resolve uncertainty too quickly, preventing learners from engaging with the material in a way that forms resilient mental models.

Ubien arises from the recognition that true understanding occurs when students do not merely receive ideas, but are made to come up with it themselves. Drawing on the principles of Inquiry-Based Learning (IBL), Ubien transforms traditional and IBL specific textbooks into interactive courses where students iteratively advance through answering sequences of questions. Instead of being told what to think, learners are socratically questioned into discovering why concepts hold together and how the structures of a discipline emerge organically from first principles.

In this environment, knowledge is not an external artifact handed down by an instructor, it becomes something the student builds. The result is a form of understanding that is permanent, self-sourced, and meaningfully tied to the learner's own cognitive efforts. Ubien seeks to restore this mode of discovery-based learning to the digital age, offering a platform where students can engage with material in a way that mirrors the curiosity-driven process that underlies real intellectual growth.

2. Requirements Details

2.1 Functional Requirements

2.1.1 System Functionality

- The system should send a confirmation email containing a verification link or code upon registration.
- The system should extract and process text from uploaded PDFs.
- The system should compute estimated token counts.

- The system should compute the estimated course creation cost.
- The system should generate course content using the ChatGPT API upon user approval.
- The system should store all generated course content (sections, questions, metadata).
- The system should track user progress (completed tasks and questions).
- The system should store token estimates and cost calculations.
- The system should generate responses using the ChatGPT API.
- When Study Mode A is selected, the system should display text-only responses.
- When Study Mode B is selected, the system should generate a neutral upper-body avatar.
- When Study Mode C is selected, the system should generate animation data including gaze direction, lip-sync motions, and an emotion tag (e.g., “happy,” “serious”).
- The system should use TTS to generate audio output.
- The system should synchronize avatar mouth movement with TTS audio.
- The system should generate animation data from LLM output (timing, gestures, expressions).
- The system should ensure TTS–animation synchronization delay of less than 150 ms.
- The system should detect invalid/unreadable PDFs and show an error.
- The system must prevent course creation when the PDF is invalid.
- The system should alert the user if the PDF exceeds the maximum token limit.
- The system should alert the user if ChatGPT or TTS API failures occur.

2.1.2 User Functionality

- The user should be able to create an account by entering the required information, including an email address, password, age, and school level.
- The user should be able to activate their account using a verification link or code sent to them via email.
- The user should be able to log in only after verifying their email address.
- The user should be able to upload any textbook in PDF format.
- The user should be able to view the computed price before creating the course.
- The user should be able to confirm or cancel the creation of a course after seeing the price.
- The user should be able to view the list of courses they have created.
- The user should be able to access any course they have created.
- The user should be able to interact with the course material by asking questions, replying to the instructor, and seeking clarifications through the system’s interface.
- The user should be able to open and close a virtual 3D instructor window with TTS functionality to chat with and listen to the course material.
- The user should be able to select between Mode A, Mode B, and Mode C as instructor types.

2.2 Non-functional Requirements

2.2.1. Usability

- The system should allow a first-time user with no technical background to upload a PDF and start a course within 2 minutes of initial use.
- All core actions, for instance, uploading textbooks, accepting the cost, selecting study modes, and asking questions, should be accessible in no more than four clicks from the dashboard.

2.2.2. Reliability

- In the event of network failures, API timeouts, or invalid uploads, the system displays an error message within 10 seconds and allows users to resubmit the request with descriptive error messages.
- The backend system will maintain an availability of at least 99% during regular operational hours.
- User progress, including completed tasks, questions asked, and lessons viewed, will be saved instantly and remain secure, even if the browser is refreshed or closed.

2.2.3. Performance

- Under typical network conditions, which include a connection with 50–100 ms latency and at least 10 Mbps download bandwidth, only text ChatGPT API responses will be delivered within 3-5 seconds.
- Once the user approves the cost, the system will initiate the course generation process within one minute.
- Text extraction and token estimation for PDFs up to 50 MB should take 10 seconds or less on the backend.
- The system will perform text summarization, content chunking, and filtering to avoid exceeding the ChatGPT API token limit or context buffer size.
- WebGL-based animations will maintain a minimum of 25 frames per second on mid-range laptops with 8 GB of RAM and an integrated GPU.

2.2.4. Supportability

- Each of the system's distinct modules, PDF parsing, token estimation, LLM processing, and animation control, can be upgraded or changed without impacting TTS, and animate the system as a whole.
- The system should record failed API requests, PDF extraction errors, and token estimation failures, along with the timestamps and request details. To facilitate maintenance and troubleshooting, logs must be retained for a minimum of 30 days.
- API keys, price values, and animation choices should all be customizable using environment variables rather than being hard-coded. This will eliminate the need for recompilation of the backend.

2.2.5. Scalability

- The backend should be able to run multiple instances simultaneously and support 200 active users without any loss of performance.
- To reduce response time by at least 50% for repeated requests, frequently reused LLM outputs, such as textbook summaries and chapter metadata, will be cached.
- The system should load the list of courses within two seconds when a user opens the dashboard, even if the database contains a large number of courses, with a total of 10,000 or more courses stored.

3. Final Architecture and Design Details

3.1 Overview

In this section, we will firstly explain the subsystem decomposition of our application where the main subsystems of the overall program can be mapped out in detail. Then we will provide the names of the subsystems, what technologies they utilize and how they connect with each other.

3.2 Subsystem Decomposition

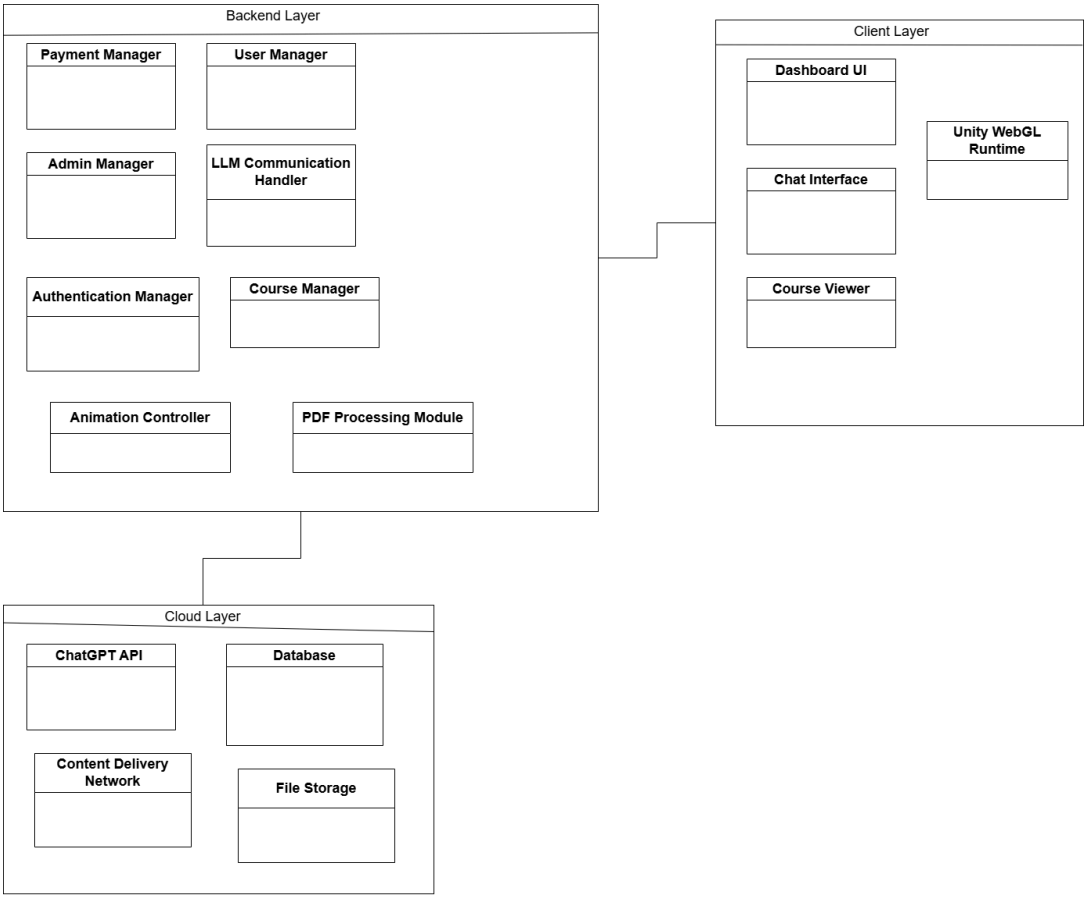

Figure 1: Subsystem decomposition diagram

Our system is formed from 3 main layers that communicate with each other but work independently. The first of these is the backend layer that handles all the payment, registration and the course creation logic of the program. The second one is the client layer that the user interacts with to navigate through the service and interact with the courses they create. And the third is the cloud layer which is used to store all the created courses and the models needed for the system to function. This includes the database and the LLM APIs that are connected for the system to work.

3.3 Backend Layer

We have decided to use Go as the core programming language and runtime for our backend layer. Go is highly efficient for server applications and provides a robust standard net/http web server model, which we utilize to listen and serve incoming requests. For routing and middleware, we have integrated the chi framework, which allows us to maintain clean and manageable REST-like HTTP endpoints. Rather than relying on a separate client-side application framework, we utilize a server-side rendering style via Go's native html/template library for dynamic view generation. This allows the project to be more easily expandable with less parts to work in the codebase.

For authentication and session management, we implemented a cookie based session approach utilizing an in-memory session store. To ensure user credentials are safe and secure, user passwords are encrypted and hashed using the bcrypt library. For asynchronous browser actions, our core API architecture handles standard JSON requests and responses, alongside multipart/form-data protocols for handling textbook PDF ingestions when a user creates a new course.

To provide a seamless user experience with real-time data, we established a WebSocket event stream via the gorilla/websocket library using a hub and broadcast model. This real-time API is essential because our heavy course creation pipeline runs asynchronously in the background. The backend continuously emits live progress updates, initialization statuses, and potential failure alerts directly to the user dashboard through this connection. Furthermore, the backend is designed to integrate external process executions, shelling out to a Python 3 script for the core PDF processing pipeline. The main purpose of the Backend Layer is to provide a highly concurrent, secure, and responsive foundation for the platform while handling intense background processing and real-time client communication efficiently.

3.4 Client Layer

The Client Layer of our system operates as a web application built upon a hybrid foundation of SSR pages and modular JavaScript. The frontend architecture is driven by HTML templates generated by the backend, which are then made fully interactive using client-side ES modules. For the user interface and styling, we utilize Tailwind CSS delivered, allowing for a responsive and scalable design approach. To safely and accurately display complex educational content which mainly includes mathematical notation, the client integrates “marked” for Markdown rendering, DOMPurify to ensure strict HTML sanitization,

and KaTeX (with auto-render functionality) for the precise typesetting of mathematical formulas.

Data communication and state management within the browser relies on a mix of modern Web APIs and protocols. Standard HTTPS is used for initial page loads, static asset serving, and core API interactions using the fetch API. For heavy file uploads, such as textbook PDF ingestions, the client utilizes XMLHttpRequest to provide the user with accurate, real-time upload progress UI. Furthermore, continuous real-time data flow is achieved through secure WebSocket connections. The client also utilizes native browser capabilities like FormData for efficient data packaging, localStorage for retaining client-side session states, and HTML5 audio elements for TTS output.

A defining feature of the Client Layer is the interactive 3D instructor subsystem. This component is powered by a Unity WebGL export embedded into the web interface via an iframe. To orchestrate smooth communication between the primary web DOM and the encapsulated Unity environment, the system employs the postMessage API. This allows the JavaScript modules to securely control the avatar's state, driving lip-sync and animation updates using requestAnimationFrame calls to ensure smoothness and synchronization.

3.5 Database Layer (Persistence)

For the data storage layer of our system, we have used PostgreSQL 16 as our primary database engine. To ensure consistency between development, testing, and production environments, the database is containerized using the postgres:16 Docker image.

When it comes to application to database communication, we made the decision to bypass object relational mapping frameworks in favor of executing direct SQL queries from the Go backend. This approach maximizes query optimization, reduces abstraction overhead, and provides more control over complex database operations. The Go application interfaces with PostgreSQL by using Go's standard database/sql package, which is powered by the pgx standard library driver bridge to handle secure and efficient connection pooling.

Schema management and versioning are handled to support safe, evolving deployments. The architectural baseline is defined in a centralized schema file, while subsequent database evolutions are tracked and executed via incremental SQL migration scripts. Furthermore, the system utilizes native PostgreSQL features to shift critical data validation closer to the storage layer. We employ table constraints and checks such as validating source_mode inputs, enforcing accurate progress ranges, and ensuring enum-like status verifications. Data lifecycles are securely managed using foreign key relationships configured with cascade deletes, while overall query performance is improved through strategically mapped indexes. Finally, to ensure auditability and precision, we implemented a custom PostgreSQL database trigger and function to automatically update updated_at timestamps whenever row modifications occur.

4. Development/Implementation Details

4.1 Client Layer

The Client Layer serves as the primary interactive interface for users engaging with the Ubien platform. From an implementation perspective, the client is not developed as a separate repository. Instead, it relies on server-rendered HTML templates combined with Tailwind CSS via CDN for styling. For displaying rich text and complex equations, the system utilizes a pipeline of marked, DOMPurify, and KaTeX. The interactivity is managed through vanilla ES modules located in the static/js directory. The client layer is divided into four main interactive components: the Dashboard UI, the Chat Interface, the Course Viewer, and the Unity WebGL Runtime.

4.1.1 Dashboard UI

The Dashboard UI acts as the central hub for users to manage their courses. The structural layout is defined within templates/dashboard.html, which features a responsive sidebar, a sticky top navigation bar, and CSS keyframe animations to enhance user experience.

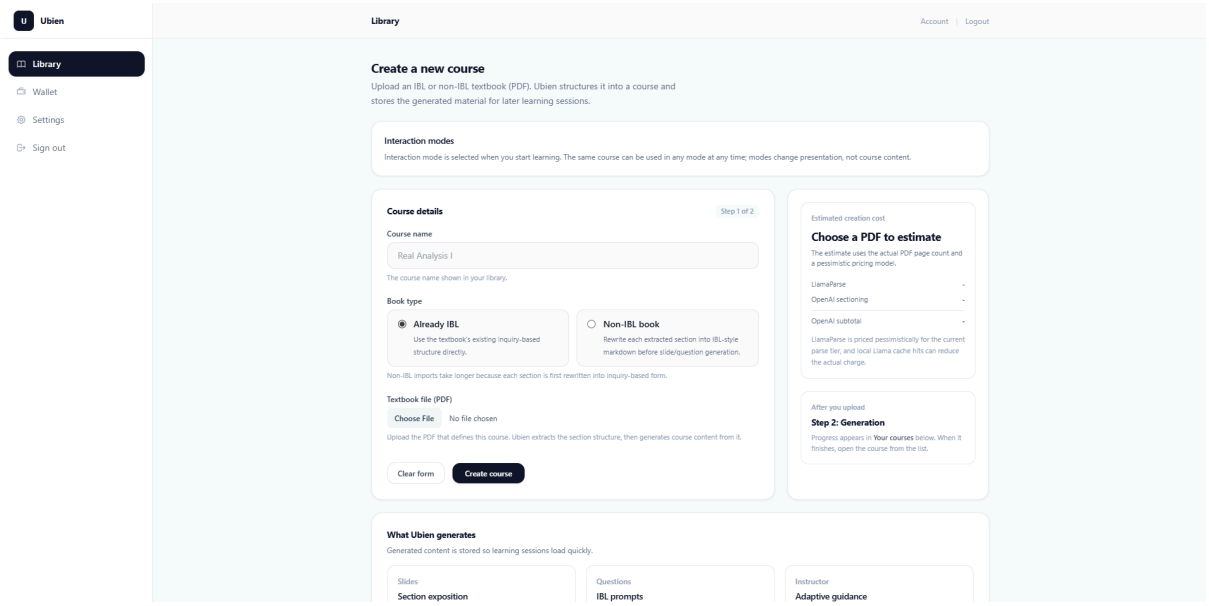

Figure 2: User dashboard of Ubien

The interactivity of the dashboard is managed by several specific JavaScript modules. Firstly the environment is created. This includes setting up upload forms, error modals, and initializing the WebSocket connections. Then the dashboard module establishes a WebSocket connection to the /ws endpoint on the server. It parses incoming JSON payloads and delegates events to specific UI rendering functions. The live course creation visual listens to the initialization, progress, and failure events emitted by the WebSocket to dynamically build and update the course list DOM. This module also handles the UX interactions, such as bulk selection, course deletion, progress resetting, and rendering safe HTML strings using an escapeHtml utility to prevent XSS attacks.

4.1.2 Chat Interface

The conversational core of the platform is implemented through two distinct chat contexts, both designed to facilitate IBL. The standalone chat provides a dedicated, single-column interface featuring a chat log and input form. User submissions are sent asynchronously via POST requests using the Fetch API. Responses from the AI assistant are rendered dynamically through a strict pipeline: Markdown is parsed via Marked, sanitized using DOMPurify, and mathematical notations are formatted using the KaTeX auto-render extension. User input bubbles remain as secure, plain text.

The In-section tutoring is integrated directly into the course workflow. This context shares the same rendering pipeline for AI responses but submits data via a structured POST request, passing the specific item_id and the message history. The responses in this context are tightly coupled with the course logic, triggering events such as reaching a correct answer, refreshing narration paths, and hooking into the TTS autoplay system to drive the avatar.

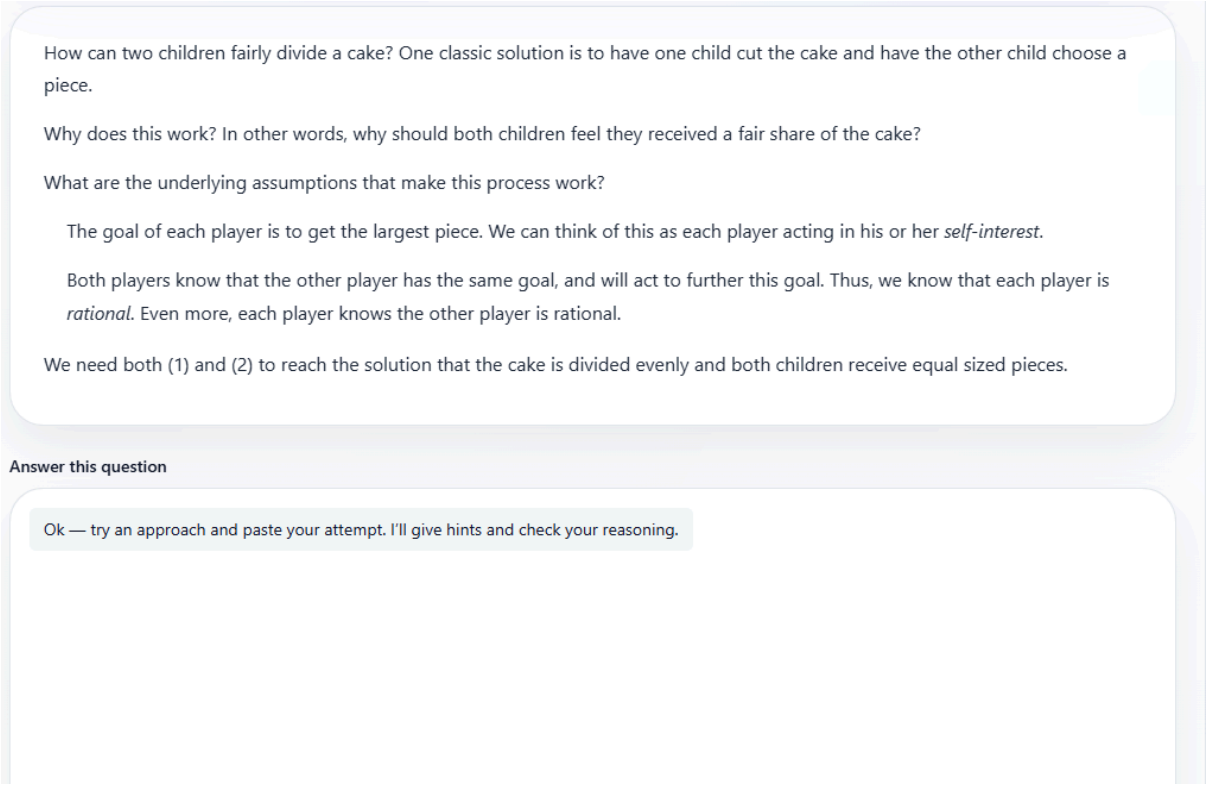

Figure 3: Chat interface with an interactive question

4.1.3 Course Viewer

The Course Viewer serves as the primary educational interface where users interact with the generated content. Its structural foundation is built on an HTML template that provides a grid layout. This layout dynamically adapts to display the virtual teacher panel either on the side or at the top, depending on the user's screen dimensions. The template also establishes the typographical rules for rendering educational materials, ensuring that elements like blockquotes, data tables, code blocks, and mathematical equations are displayed clearly. A dedicated configuration script is embedded within this template to manage the initial state, hold essential layout constants, and establish the connection points for the embedded 3D avatar.

The core interactivity of the viewer is driven by a runtime logic script that governs the sequence of educational items, which distinguishes between informative slides and interactive questions. This script manages user navigation and handles complex interactions such as custom keyboard shortcuts, undo actions, and tracking completion milestones. It transmits the progress telemetry to the backend without interrupting the user's workflow, while also orchestrating the play, pause, and restart controls for the audio narration. To ensure the educational content is rendered accurately, a separate processing script manages the text pipeline. This pipeline applies custom heuristics to protect inline mathematical notation from being misinterpreted, such as distinguishing regular currency symbols from equation delimiters, before passing the text through the platform's markdown parser and mathematical typesetting libraries.

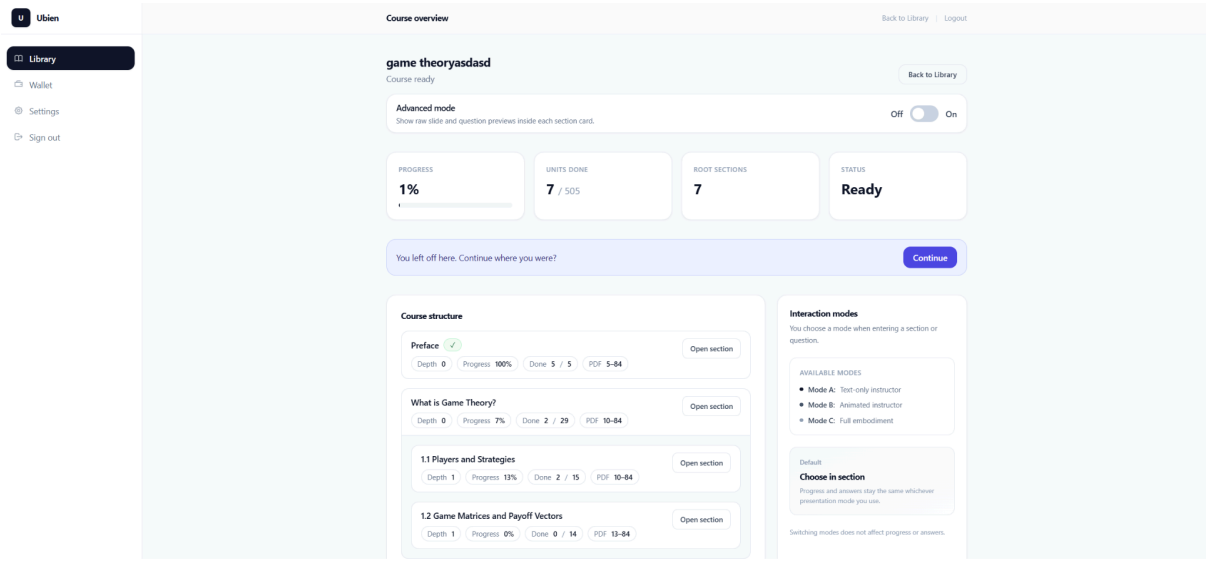

Figure 4: Course viewer page of an IBL book

4.1.4 Unity WebGL Runtime

To provide an immersive learning environment, the system integrates a 3D avatar powered by a WebGL application. The 3D environment runs autonomously within a dedicated, isolated iframe. The integration relies on an intermediary host file inside the iframe that manages the standard WebGL initialization process. This host is responsible for loading the compiled WebAssembly files and necessary framework assets, displaying initial loading progress bars, and automatically resizing the 3D canvas to match the user's viewing container, regardless of whether they are on a mobile or desktop device.

Communication between the main web application and the isolated 3D environment is handled through a secure messaging bridge. The iframe host listens for specific event messages dispatched from the parent interface and securely forwards them to a designated controller object within the 3D engine. This setup allows the main web application to dynamically dictate avatar parameters, such as changing character models, triggering idle or explanatory animations, and feeding precise lip-sync data streams to the character in real time. On the parent application side, a primary integration script acts as the master controller. It ensures that no commands are sent until the 3D engine explicitly signals that it is fully loaded and ready, safely queuing any necessary actions in the meantime. This integration layer manages a comprehensive catalog of avatar body styles and facial presets

while tightly synchronizing the avatar's visual speaking state with the platform's active audio narration.

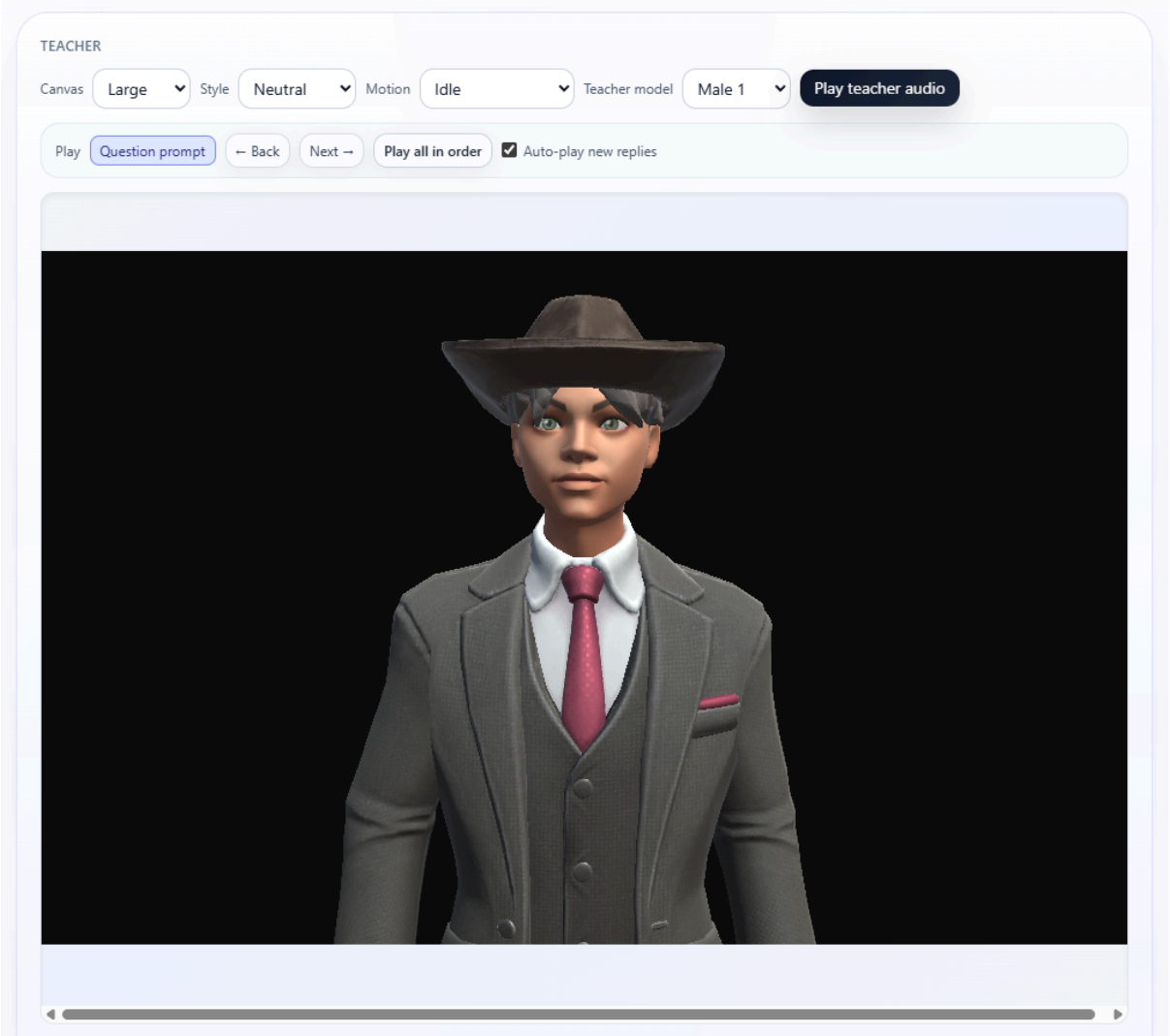

Figure 5: Enabled 3D instructor with TTS functionality on the course page

4.2 Backend Layer

4.2.1 Functionality Managers

The core administrative and transactional logic of the platform is consolidated within a set of dedicated functionality managers. The financial lifecycle is governed by a payment manager modeled in micro-USD. It operates on a robust reservation system that debits a user's wallet upfront based on estimated costs for upcoming LLM and PDF parsing services. Once a task successfully completes or fails, a settlement routine reconciles the actual token usage, automatically issuing partial refunds or full releases as necessary. This manager also maintains the overarching platform ledger, providing accessible data views for balance summaries, charge breakdowns, and subscription updates via standard HTTP endpoints. User identity and profile management are handled directly through the authentication and

dashboard routes. This includes securely registering users with bcrypt password hashes and managing account updates guarded by strict current-password verification protocols.

Furthermore, the platform's educational content is orchestrated by a course manager that dictates the complete lifecycle of a textbook's transformation. It handles the initial PDF upload, stores the file locally, asynchronously triggers the background creation pipeline, and actively pushes pipeline state updates through a WebSocket hub directly to the client dashboard. It also manages the retrieval of generated course trees and persistently tracks user progress through the interactive slides and questions.

4.2.2 LLM Communication Handler

The LLM Communication Handler serves as the bridge between the Ubien platform and external generative AI services, systematically managing prompt engineering, response parsing, and cost attribution. This functionality is divided across two primary operational domains. For interactive learning, the runtime web component manages both the standalone chat and the in-section tutoring features. It utilizes thin billing wrappers that reserve user wallet funds, invoke the openai-go client, calculate the exact usage tokens upon receiving a response, and subsequently settle the financial reservation. On the other hand, during the asynchronous PDF processing phase, a mirrored billing wrapper is utilized for the course generation pipeline. In this context, the handler builds complex, domain-specific prompts designed to accurately extract textbook structures, appropriately format markdown, and safely parse the AI's outputs into internal Inquiry-Based Learning JSON structures, which are necessary for generating the final interactive educational content.

4.2.3 Animation Controller

The Animation Controller orchestrates the synchronized delivery of speech, narration, and avatar state data required by the Unity WebGL client. When a user requests narration or interacts with the virtual instructor, the controller resolves the required spoken text and invokes AWS Polly twice: once to generate an MP3 audio stream, and again to generate JSON speech marks for precise viseme extraction. The controller packages this audio data alongside authored timeline states (including mode, gender, emotion tags, and animation hints) into a JSON payload. This comprehensive render package is then delivered to the client, providing the exact timing and synchronization hooks needed to drive the interactive 3D avatar within the browser iframe.

4.2.4 PDF Processing Pipeline

4.2.4.1 PDF Pre-Processing

The PDF pre-processing stage is the first major part of the overall pipeline, and its purpose is to transform an uploaded textbook PDF into a structured representation that the rest of the system can use. Once course creation begins, the backend resolves the local path of the uploaded PDF and performs an initial validation step by calling qpdf to determine the number of pages. That page count is used both as a sanity check and as an input to billing, because the parsing stage relies on LlamaParse, an external paid document-processing service. Before the parse starts, the backend checks whether a usable local LlamaParse cache already exists for that exact PDF; if not, it reserves the expected provider cost through the

wallet/payment subsystem. The backend then launches a dedicated Python pipeline script and passes it the PDF path, output locations, parse tier/version settings, and a job identifier. While the script runs, it emits structured JSONL progress events such as stage changes, warnings, errors, and completion notices. The Go backend reads these events in real time, maps them onto course-creation progress percentages, logs them, and forwards the progress state to the user dashboard.

Inside the Python pipeline, the PDF first goes through LlamaParse to produce full-document markdown, page-by-page markdown, parse metadata, and cached image assets. From there, the pipeline tries to recover the document's structure as faithfully as possible. If the PDF already contains a native outline or bookmark tree, that is used as the table of contents source; otherwise, the pipeline identifies an explicit contents section from the parsed markdown, isolates that TOC window, and removes it from the body text. It then extracts headings from the reconstructed body markdown, cleans and merges noisy heading fragments, serializes the TOC into a hierarchical list of entries, and matches the cleaned body headings back to those TOC entries. This alignment step is what allows the system to rebuild the book as a navigable tree of sections, each with its own title, text span, page information, and any associated figures. After the tree is built, the pipeline performs additional content repairs, including attaching cropped figures and downloaded page images, repairing Mermaid diagram blocks, compiling them into SVG where possible, and fixing malformed mathematical notation for MathJax rendering. The final result is written out as a structured JSON tree plus companion artifacts such as a text tree and static site export.

4.2.4.2 Course Generation

Once PDF pre-processing is complete, the backend reads the structured JSON tree and converts each parsed node into the system's internal CourseSection representation. This conversion preserves the section hierarchy, titles, extracted source text, page boundaries, child sections, and figure references, effectively turning the parsed textbook into a structured course outline. At this stage the system has moved from document understanding to course assembly. If the uploaded source is marked as a non-IBL text, the backend applies extra transformation steps before continuing: it selects an appropriate section granularity, filters out front matter that is not instructionally useful, and rewrites the remaining sections into a more inquiry-based learning format so they are better suited to interactive teaching.

After the section structure has been finalized, the backend stores the sections in the database and begins generating the learner-facing course content for each one. For every stored section with usable text, the system calls downstream LLM-based prompts that transform the source section into instructional items such as explanatory slides and tutor-style question blocks. These generated items are then written back into the database as the actual content presented later in the course interface. The complete course generation pipeline therefore proceeds in two linked phases: PDF pre-processing first converts the uploaded textbook into a cleaned, structured, media-aware tree of sections, and the course generation stage then converts that tree into stored sections and interactive learning items. Together, these stages transform an unstructured textbook PDF into a usable digital course that can be delivered through the platform.

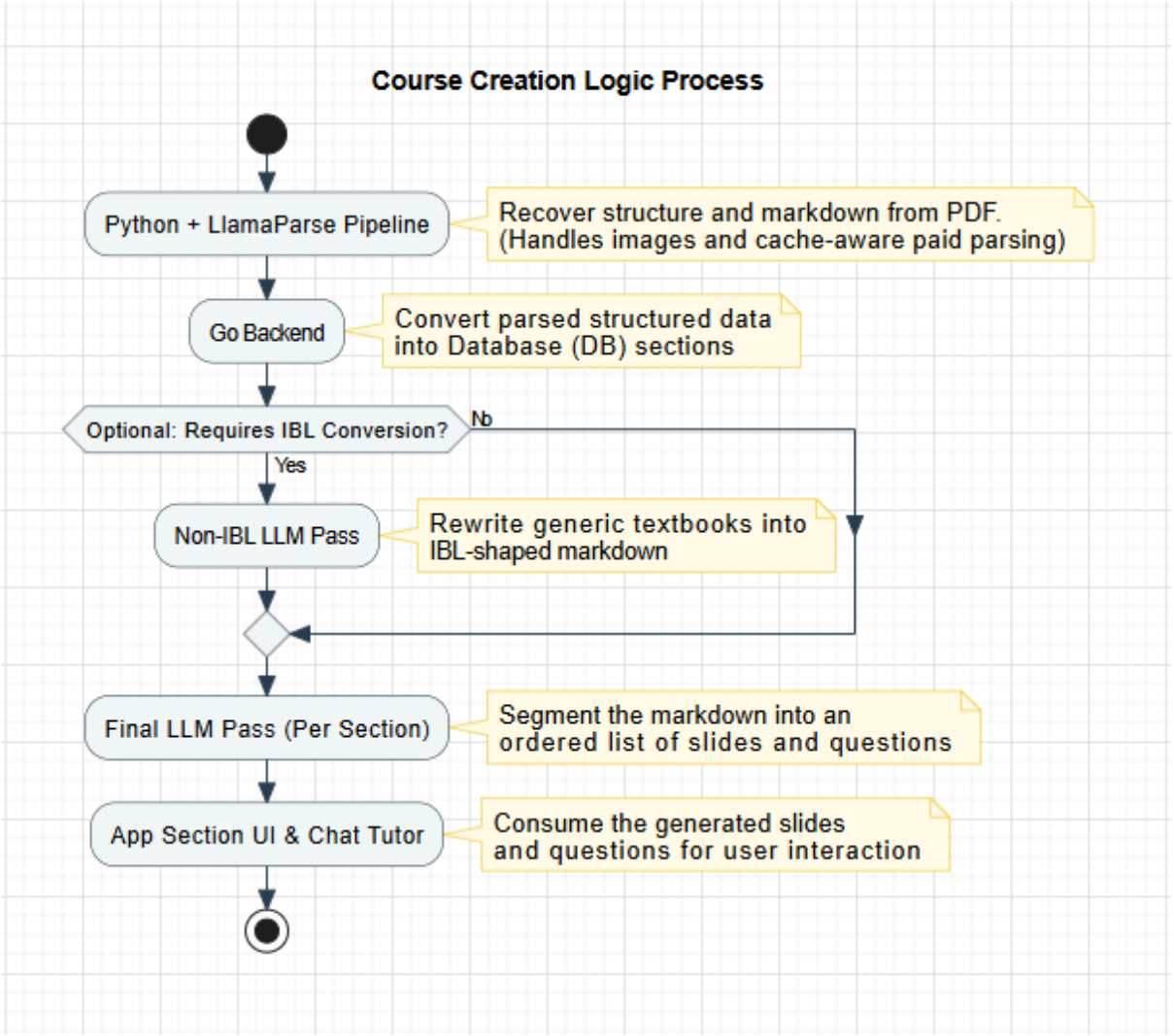

Figure 6: Flow chart of the course creation pipeline

4.3 Cloud Layer

The cloud and infrastructure layer of the Ubien platform operates on a containerized architecture that utilizes Docker Compose. The core environment consists of a primary web service and a PostgreSQL. The main web application image is built directly from the project's Dockerfile and is configured with a strict dependency on the database container to ensure proper initialization. For development efficiency, the system utilizes bind-mounted source directories and relies on the air command for live-reloading. The architecture deliberately avoids overly complex orchestration layers, such as Kubernetes, opting instead for a highly efficient containerized setup where access to external services is injected securely into the web container via environment variables.

To fulfill the platform's generative capabilities, the cloud layer integrates with several third-party SaaS APIs. The ChatGPT API serves as the primary generative engine, handling conversational responses, course section structuring, and narration scripting. All interactions with the OpenAI API are wrapped within the backend's internal billing logic to monitor token usage and reserve funds dynamically. For the initial textbook processing, the system connects to Llama Cloud, leveraging its API within the Python pipeline to handle PDF ingestion and markdown layout parsing, which is billed per parse unless cached locally.

Furthermore, the platform integrates with AWS for the TTS functionality needed for the 3D instructor.

5. Test Cases and Results

5.1 Functional Test Cases

Test ID: 1
Category: Functional
Severity: Critical
Objective
Verify that a user can successfully register in the system.
Steps

1. Open the application in a web browser.
2. Click the Register button.
3. Enter valid email, username, and password.
4. Click Submit.

Expected Result
The system successfully creates a user account and sends a verification email.

Date-Result: **01.05.2026 - Successfull**

Test ID: 2
Category: Functional
Severity: Major
Objective
Verify that users cannot log in with incorrect credentials.
Steps

1. Open the login page.
2. Enter a valid email and an incorrect password.
3. Click Login.

Expected Result
The system displays an Invalid credentials message and denies access.

Date-Result: **01.05.2026 - Successfull**

Test ID: 3
Category: Functional
Severity: Critical
Objective
Verify that a verified user can successfully log in.
Steps

1. Navigate to the login page.
2. Enter valid credentials.
3. Click Login.

Expected Result
The user is redirected to the Dashboard page.

Date-Result: **01.05.2026 - Successfull**

Test ID: 4
Category: Functional
Severity: Major
Objective
Verify that users can successfully log out.
Steps

1. Log into the system.
2. Click the Logout button.

Expected Result
The system logs the user out and redirects to the login page.

Date-Result: **01.05.2026 - Successfull**

Test ID: 5
Category: Functional
Severity: Major
Objective
Verify that the dashboard displays all user courses.
Steps

1. Login to the system.
2. Navigate to the Dashboard page.

Expected Result
All previously created courses appear with title, creation date, and progress.

Date-Result: 01.05.2026 - Successfull

Test ID: 6
Category: Functional
Severity: Major
Objective
Verify that a user can create a new course.
Steps

1. Log into the system.
2. Navigate to the Dashboard.
3. Click Create Course.
4. Upload a valid PDF textbook.
5. Submit the form.

Expected Result
The system successfully creates a new course entry.

Date-Result: 01.05.2026 - Successfull

Test ID: 7
Category: Functional
Severity: Minor
Objective
Verify that the course progress status is displayed correctly.
Steps

1. Log into the system.
2. Navigate to the Dashboard.
3. Observe the progress indicator for each course.

Expected Result
Each course shows correct progress information.

Date-Result: **01.05.2026 - Successfull**

Test ID: 8
Category:Functional
Severity: Critical
Objective
Verify that users can upload a valid PDF file.
Steps

1. Log into the system.
2. Click Create Course.
3. Upload a valid PDF file.
4. Click Submit.

Expected Result
The PDF file is uploaded and processed by the backend.

Date-Result: **01.05.2026 - Successfull**

Test ID: 9
Category: Functional
Severity: Major
Objective
Ensure the system rejects unsupported file types.
Steps

1. Log into the system.
2. Click Create Course.
3. Attempt to upload a .docx file.

Expected Result
The system rejects the upload.

Date-Result: **01.05.2026 - Successfull**

Test ID: 10
Category: Functional
Severity: Major
Objective
Verify that the system extracts text from the uploaded PDF.
Steps

1. Upload a valid PDF file.
2. Wait for the processing to complete.

Expected Result
Text is successfully extracted and sent to the AI service.

Date-Result: 01.05.2026 - Successfull

Test ID: 11
Category: Functional
Severity: Critical
Objective
Verify that the system generates an IBL course from a PDF.
Steps

1. Upload a valid textbook PDF.
2. Wait for course generation.
3. Open the Course Viewer.

Expected Result
The system generates structured course content.

Date-Result: 01.05.2026 - Successfull

Test ID: 12
Category: Functional
Severity: Major
Objective
Verify that the AI generates meaningful questions from course content.
Steps

1. Generate a course from a PDF.
2. Navigate to the Course Viewer.

Expected Result
The system displays meaningful questions based on the course material.

Date-Result: **01.05.2026 - Successfull**

Test ID: 13
Category: Functional
Severity: Major
Objective
Verify system behavior if AI fails to generate course content.
Steps

1. Upload a PDF with minimal or unreadable text.
2. Start course generation.

Expected Result
The system shows an appropriate error message.

Date-Result: **01.05.2026 - Successfull**

Test ID: 14
Category: Functional
Severity: Major
Objective
Verify that users can send messages to the AI instructor.
Steps

1. Open the Chat Interface.
2. Type a question.
3. Click Send.

Expected Result
The AI instructor returns a relevant response.

Date-Result: **01.05.2026 - Successfull**

Test ID: 15
Category: Functional
Severity: Major
Objective

Verify that chat history is preserved during a session.
Steps

1. Send multiple messages in the chat interface.

2. Scroll through the chat history.

Expected Result
Previous messages remain visible.

Date-Result: **01.05.2026 - Successfull**

Test ID: 16
Category: Functional
Severity: Major
Objective
Verify that users can request clarification from the AI instructor.
Steps

1. Ask a question in the chat.

2. Ask a follow-up question.

Expected Result
The AI responds contextually based on the previous message.

Date-Result: **01.05.2026 - Successfull**

Test ID: 17
Category: Functional
Severity: Major
Objective
Verify that generated course slides are displayed correctly.
Steps

1. Open a generated course.

2. Navigate through course sections.

Expected Result
Slides display correctly.

Date-Result: **01.05.2026 - Successfull**

Test ID: 18
Category: Functional
Severity: Major
Objective
Verify that users can answer course questions.
Steps

1. Open a course section.

2. Select an answer to a question.

Expected Result
The system records the answer.

Date-Result: 01.05.2026 - Successfull

Test ID: 19
Category: Functional
Severity: Major
Objective
Verify that course progress is updated after answering questions.
Steps

1. Complete a question in the course viewer.

Expected Result
Progress indicator updates accordingly.

Date-Result: 01.05.2026 - Successfull

Test ID: 20
Category: Functional
Severity: Major
Objective
Verify that the instructor avatar appears in the interface.
Steps

1. Open the Course Viewer.

2. Observe the instructor avatar.

Expected Result
The avatar loads successfully.

Date-Result: **01.05.2026 - Successfull**

Test ID: 21
Category: Functional
Severity: Major
Objective
Verify that the avatar lip-syncs with generated speech.
Steps

1. Ask a question to the AI instructor.

2. Wait for the spoken response.

Expected Result
Lip movements match the speech.

Date-Result: **01.05.2026 - Successfull**

Test ID: 22
Category: Functional
Severity: Major
Objective
Verify that the avatar performs gestures during explanations.
Steps

1. Ask a question that triggers a long explanation.

Expected Result
The avatar performs gestures and expressions.

Date-Result: **01.05.2026 - Successfull**

Test ID: 23
Category: Functional
Severity: Major
Objective
Verify profile information can be updated successfully.
Steps

1. Navigate to Profile Page.

2. Click “Edit”.

3. Change profile details.

4. Click “Save”

Expected Result
Profile is updated and a success message appears.

Date-Result: **01.05.2026 - Successfull**

Test ID: 24
Category: Functional
Severity: Major
Objective
Verify error is shown when profile update fields are left empty.
Steps

1. Navigate to Profile Page.
2. Click “Edit”.
3. Leave all fields empty.
4. Click “Save”.

Expected Result
Profile is not updated and an error message appears.

Date-Result: **01.05.2026 - Successfull**

Test ID: 25
Category: Functional
Severity: Major

Objective
Verify that a user can successfully subscribe to a plan.

Steps

1. Navigate to Subscription page.
2. Select a subscription plan.
3. Enter payment details.
4. Confirm subscription.

Expected Result
The user’s subscription is activated and reflected in their account.

Date-Result: **01.05.2026 - Successfull**

Test ID: 26
Category: Functional
Severity: Major

Objective
Verify that subscription fails when payment is declined.

Steps

1. Navigate to Subscription page.
2. Select a subscription plan.
3. Enter invalid or declined payment details.
4. Confirm subscription.

Expected Result
Subscription is not activated and an error message is displayed.

Date-Result: **01.05.2026 - Successfull**

Test ID: 27
Category: Functional
Severity: Critical
Objective
Verify AI-generated question creation.
Steps

1. Navigate to Course Creation page.
2. Upload source material.
3. Click “Generate Course”.

Expected Result
AI-generated questions is created and displayed successfully.

Date-Result: **01.05.2026 - Successfull**

Test ID: 28
Category: Functional
Severity: Major

Objective
Verify that user progress is saved correctly in the database.

Steps

1. Open a course and answer several questions.
2. Log out.
3. Log in again and reopen the same course section.

Expected Result
All previously completed questions and slide positions are restored correctly.

Date-Result: 01.05.2026 - Successfull

Test ID: 29
Category: Functional
Severity: Major

Objective
Verify that course creation is blocked when quota is exhausted.

Steps

1. Navigate to Course Creation page.
2. Attempt to create a new course with quota exceeded.

Expected Result
Course creation is blocked and an appropriate warning is displayed.

Date-Result: 01.05.2026 - Successfull

Test ID: 30
Category: Functional
Severity: Critical
Objective
Verify AI answers adapt to user-provided context.
Steps

1. Provide context or background info to AI.

2. Ask a related question.

Expected Result
AI response reflects the provided context accurately.

Date-Result: 01.05.2026 - Successfull

Test ID: 31
Category: Functional
Severity: Major

Objective
Verify that the user can access course content.

Steps

1. Navigate to the Dashboard.

2. Select a course.

3. Open a chapter and section.

Expected Result
The selected course content is displayed correctly.

Date-Result: 01.05.2026 - Successfull

Test ID: 32
Category: Functional
Severity: Major

Objective
Verify that a user can navigate slides within a section.

Steps

1. Open a course section with slides.

2. Navigate forward and backward between slides.

Expected Result
Slides are displayed correctly and progress is tracked.

Date-Result: **01.05.2026 - Successfull**

Test ID: 33
Category: Functional
Severity: Major
Objective
Verify that progress is correctly updated when a user completes a section.
Steps

1. Complete all slides and questions in a section.

2. Check the progress view.

Expected Result
The completed section is marked correctly in the progress dashboard.

Date-Result: **01.05.2026 - Successfull**

Test ID: 34
Category: Functional
Severity: Major
Objective
Verify that a user's course creation quota is not decremented if course generation fails.
Steps

1. Start creating a course.

2. Simulate failure in the AI service during generation.

3. Check the user's remaining course quota.
   Expected Result
   The monthly course creation quota remains unchanged if course generation fails.

Date-Result: **01.05.2026 - Successfull**

5.2 Non-Functional Test Cases

Test ID: 35
Category: Performance
Severity: Critical
Objective

Verify that AI responses are generated within acceptable time.
Steps

1. Ask a question in the chat interface.

2. Measure response time.

Expected Result
The response is generated within 5 seconds.

Date-Result: 01.05.2026 - Successfull

Test ID: 36
Category: Performance
Severity: Major
Objective
Verify system performance during multiple simultaneous users.
Steps
1. Simulate 50 users interacting with the system simultaneously.

Expected Result
System performance remains stable.

Date-Result: 01.05.2026 - Successfull

Test ID: 37
Category: Security
Severity: Critical
Objective
Verify that unauthorized users cannot access course data.
Steps
1. Attempt to access another user's course via URL.

Expected Result
Access is denied.

Date-Result: 01.05.2026 - Successfull

Test ID: 38
Category: Security
Severity: Critical
Objective
Verify that user passwords are securely stored.
Steps
1. Inspect database entries.

Expected Result
Passwords are stored in encrypted/hashed format.

Date-Result: 01.05.2026 - Successfull

Test ID: 39
Category: Reliability
Severity: Major
Objective
Verify system stability if AI service is temporarily unavailable.
Steps
1. Simulate AI API failure.

Expected Result
System displays a proper error message.

Date-Result: 01.05.2026 - Successfull

Test ID: 40
Category: Compatibility
Severity: Major
Objective
Verify cross-browser compatibility.
Steps
1. Run the application in Chrome, Firefox, Safari, and Edge.

Expected Result
Application functions correctly in all browsers.

Date-Result: 01.05.2026 - Successfull

Test ID: 41
Category: Usability
Severity: Minor
Objective
Verify that the interface is understandable for new users.
Steps
1. Conduct a usability test with new users.

Expected Result
Users can complete basic tasks without assistance.

Date-Result: 01.05.2026 - Successfull

Test ID: 42
Category: Reliability
Severity: Major

Objective
Verify that the system saves course progress correctly.
Steps

1. Answer several course questions.

2. Log out and log in again.

Expected Result
Progress is preserved.

Date-Result: 01.05.2026 - Successfull

Test ID: 43
Category: Performance
Severity: Minor
Objective
Verify login operation completes in less than 2 seconds.
Steps
1. Enter credentials.

2. Click “Sign In”.

Expected Result
Login operation completes in <2 seconds.

Date-Result: 01.05.2026 - Successfull

Test ID: 44
Category: Performance
Severity: Minor
Objective
Verify AI course generation completes in less than 10 minutes.
Steps

1. Upload PDF.

2. Request AI course generation.

Expected Result
Generated course appears in <10 minutes.

Date-Result: 01.05.2026 - Successfull

Test ID: 45
Category: Performance
Severity: Minor
Objective
Verify loading indicator appears while fetching AI course data.
Steps

1. Navigate to course generation page.

Expected Result
Loading icon is shown while fetching data .

Date-Result: 01.05.2026 - Successfull

Test ID: 46
Category: Performance
Severity: Minor
Objective
Verify course interface loads within 5 seconds.
Steps

1. Navigate to courses
2. Select a course.

Expected Result
Page fully loads and ready in ≤5 seconds.

Date-Result: 01.05.2026 - Successfull

Test ID: 47
Category: Security
Severity: Major
Objective
Verify that unauthorized users cannot access subscription-protected courses.
Steps

1. Attempt to open a course without an active subscription.

Expected Result
The system blocks access and displays a subscription-required message.

Date-Result: 01.05.2026 - Successfull

Test ID: 48
Category: Reliability
Severity: Major
Objective
Verify that the Ubien web app is accessible through the official website.
Steps

1. Open a web browser.
2. Navigate to the Ubien web application URL.
3. Attempt to log in or access the dashboard.

Expected Result
The web app loads successfully and is fully accessible to authorized users.

Date-Result: **01.05.2026 - Successfull**

Test ID: 49
Category: Compatibility
Severity: Major
Objective
Verify that the Ubien web app interface scales correctly on different device screen sizes.
Steps

1. Open the Ubien web app on devices of various screen sizes (desktop, tablet, mobile).
2. Navigate through key pages such as dashboard, course view, and settings.
3. Check for visibility and accessibility of all text and controls.

Expected Result
The UI elements are correctly scaled and fully visible on all tested devices without overlap or cut-off text.

Date-Result: **01.05.2026 - Successfull**

6. Maintenance Plan and Details

Maintenance for the Ubien platform will be structured around managing system limitations, monitoring external API dependencies, and continuously integrating new features based on

user feedback. To ensure the continuous and reliable operation of the platform, the following maintenance strategies have been established.

6.1 Limitations of the Server and External API Dependencies

The platform's architecture depends on both internal server capacities and external third-party services, requiring strict monitoring protocols.

The Go backend and PostgreSQL database must be continuously monitored for memory consumption and bandwidth utilization when handling multiple simultaneous users during heavy background course generation processes. Scaling up server resources will be evaluated based on user growth to prevent database crashes or out-of-memory errors.

The system heavily relies on third-party APIs, specifically ChatGPT and Llama for LLM processing, and Amazon Polly for (TTS) generation. A critical maintenance routine involves strictly monitoring the account balances, billing thresholds, and overall token counts for these services. Since course generation is a highly token-intensive process, automated alerts are configured to notify administrators before account balances are depleted or strict rate limits are reached. In these situations, administrators are expected to handle the situations on a case by case basis.

In addition to financial monitoring, the system will handle API timeouts, rate limit exceedances, and temporary service outages gracefully. Fallback mechanisms and detailed logging will be maintained to ensure the platform remains stable even in the case of an external API disruption.

6.2 Version Checking and Source Control

Throughout the ongoing lifecycle of the project, GitHub will be utilized to share source code, manage collaborative development, and maintain strict version checking for all system updates. Because the system consists of distinct, independent modules which are the Go backend, the JavaScript client layer, and the Unity WebGL elements, assets versioning must be handled carefully. Updates to the backend REST API or WebSocket structures will require synchronized updates to the client layer to ensure that modifications in one area do not break the overall system's functionality.

6.3 Change Requests and Issue Tracking

User feedback and automated error logs are vital for identifying bugs. Specifically, the system will track PDF parsing extraction errors, token estimation failures during course creation, and TTS-animation synchronization delays in the 3D instructor interface. These logs will be retained for a minimum of 30 days to facilitate troubleshooting. Change requests, feature enhancements, and bug reports gathered from end users will be handled by the development team on a case by case basis. Future iterations of the platform will be planned according to these requests and developed in structured agile sprints as new release versions.

6.4 Maintaining the PDF and LLM Pipelines

The background course generation pipeline relies heavily on external process executions. Routine maintenance will involve keeping the Python processing scripts and the qpdf integration secure, optimized, and up to date with the latest library versions.

In order to ensure cost efficiency and prevent failures, the system's text summarization, content chunking, and filtering processes will be continually refined. This ongoing maintenance ensures that extracted PDF data does not exceed the maximum token limits or context buffer sizes of the ChatGPT and Llama APIs. In the case of a price hike or a change to these external systems, the development team will pivot to different models for the LLM processing or do fixes to keep the course creation price and quality stable.

6.5 3D Avatar and Animation Maintenance

The Unity WebGL runtime powering the virtual 3D instructor will be periodically profiled and optimized. The maintenance goal is to ensure the avatar subsystem maintains a minimum performance standard of 25 frames per second on mid-range devices without causing memory leaks in the browser. Efficiency issues within the animation controller, specifically regarding gaze direction, gesture timing, and the less-than-150ms delay requirement for lip-syncing with Amazon Polly TTS data, will be continuously monitored and patched to prevent a degraded end user experience.

Further improvements to the 3D models are to be continually passed out to improve the design and human-like behaviour of the instructor. This may also involve new animation cycles, idle animations, revamped models or newer TTS models with more human-like audio.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

7.1.1 Constraints

Economic and Resource Constraints: The platform relies heavily on third party services. A major constraint is managing the financial cost associated with token limits during the ingestion of large PDF textbooks and the generation of interactive course content. The main course creation logic is the major reason behind this created course with other systems like the PDF slicing system for different slides, the Amazon Polly TTS services and the ChatGPT API used for interactive questions in the courses coming in as a recognizable cost after the main PDF parsing part. With all these costs in mind, the project was developed to handle documents of different sizes but there are alerts to stop the user from using too many tokens at once to create courses or overutilizing the services.

Performance Constraints: The integration of real-time 3D elements dictates strict performance boundaries. The Unity WebGL based animations are constrained by the requirement to maintain a minimum of 25 frames per second on mid-range systems. Furthermore, to maintain a realistic interaction, the system is constrained by a strict

synchronization requirement, allowing less than 150 ms of delay between the Amazon Polly TTS audio and the avatar's lip sync animation. These constraints required us to use less detailed low-poly models during the development of the product.

Technical and Contextual Constraints: Processing textbooks is constrained by the maximum context buffer size of the utilized Large Language Models. This forces the system to utilize text summarization and content chunking to prevent API failures or token exceedances. This was done through a multi-step PDF slicing pipeline that goes over the material multiple times to select slicing points and inputting smaller slices to the LLM for a better course generation.

7.1.2 Standards

Web and Communication Standards: The client and backend communication adheres to standard RESTful HTTP(S) protocols and secure WebSockets (WSS) for real-time dashboard updates. The frontend rendering follows W3C standards (HTML5, CSS, WebGL) to ensure cross browser compatibility.

Security Standards: Password management adheres to industry-standard cryptographic practices by utilizing bcrypt for secure hashing. User information is kept in an encrypted manner in the database that is not accessible to the development team.

Accessibility Standards: By providing multiple study modes (Mode A for text-only, Modes B and C for visual/audio avatars), the platform aligns with web accessibility guidelines, accommodating users with different learning preferences or reading disabilities through integrated TTS.

Pedagogical Standards: The core functional logic of the AI adheres to the established principles of IBL, ensuring the system acts as a Socratic guide rather than a standard direct answer search engine. It is outlined in the website that this tool is not to be used as a complete replacement of other learning methods but as a supplement to it as an individualized learning tool.

7.2. Ethics and Professional Responsibilities

Data Security and Privacy: As an educational platform processing user information and proprietary textbook data, safeguarding privacy is of upmost priority. We take data security seriously by strictly enforcing protection measures, including the encryption and secure hashing of all user passwords using the bcrypt algorithm. This ensures that user credentials and personal account details remain completely secure and inaccessible to unauthorized parties.

Copyright and Intellectual Property: The platform operates by extracting and processing text from user-uploaded PDF textbooks. Because these educational materials are often protected by copyright laws, Ubien strictly prohibits course sharing among users. All generated courses and interactive materials are intended exclusively for personal use. Restricting the platform to individual, private access ensures that the system does not violate intellectual property rights or facilitate the unauthorized public distribution of copyrighted materials. The user is responsible for uploading material that is legally obtained and in agreement to being processed by LLMs.

LLM Limitations and Educational Integrity: While the system utilizes advanced Large Language Models to generate questions and responses, it is a fundamental professional responsibility to acknowledge that generative AI is not 100% accurate. Because the system

carries the inherent risk of AI hallucinations or generating misleading explanations, users must be aware that double-checking the AI's guidance against the original source material is strictly required. The AI is engineered to act as an interactive guide, not an infallible oracle. For this reason a reporting button is added to courses to report potentially wrong information created by the LLMs. This allows the development team to improve the course creation pipeline to see fewer mistakes overall.

The Supplementary Role of AI in Education: Finally, there is an ethical obligation to clearly define the platform's role within a student's broader educational journey. While Ubien is designed to cultivate deep, resilient understanding through Inquiry-Based Learning, this tool is strictly a supplementary study aid. It is absolutely not a replacement for classic learning environments, traditional schooling, or human educators. The platform serves to reinforce knowledge through active engagement, rather than replace the foundational structures of formal education.

All these aforementioned ethical responsibilities and constraints are outlined in the official Ubien website in the easily accessible terms of use page.

7.3. Teamwork Details

7.3.1. Contributing and functioning effectively on the team to establish goals, plan tasks, and meet objectives

To ensure the successful delivery of the Ubien platform, our team operated under a structured timeline. Goals were established early in the project lifecycle by breaking down the platform's complex architecture into manageable, modular tasks, such as backend API development, LLM prompt engineering, PDF processing pipelines, and Unity WebGL integration. By utilizing agile methodologies and consistent progress tracking, our team effectively delegated responsibilities according to individual technical strengths and academic interests. Still, every member helped out each other in different parts of the project to get a better overall understanding of the systems and to improve their skills. With this in mind, the main features each team member worked on is as follows:

Ahmet Deniz Gelir: Worked mostly on the course creation pipeline. Implemented and fine tuned the multi-layered PDF processing logic.

Efe Can Tatar: Worked on the 3D instructor animation and rendering parts. Took charge in project planning, management of timelines and report writing.

Kemal Onur Özkan: Worked on the front-end design and user experience of the platform.
Eren Uslu: Worked on feature testing and development of the PDF creation logic.
Simay Uygur: Worked on the animation and TTS logic of the 3D instructor. Improved the frontend design and course creation logic.

7.3.2. Helping creating a collaborative and inclusive environment

As a team we established a culture where every member was actively encouraged to share insights, propose architectural solutions, and voice concerns during design discussions. By maintaining mutual respect, actively listening to differing technical perspectives, and collectively supporting one another through complex debugging and testing phases, the team cultivated a constructive atmosphere. This environment ensured that decisions regarding system design and pedagogical implementation were made democratically, ultimately elevating the quality of the software by valuing every member's unique contributions. Regular meetings were planned to listen to every team member’s concerns and constraints and planning was done with these in mind.

7.3.3. Taking lead role and sharing leadership on the team

The team did not have a classical leader role and shared the leadership among every member with regard to every member’s schedule outside the project. If any member was busy with other tasks, the leadership was assumed by the member that was most actively working on the project at the time. This allowed us to move swiftly during development and accommodate each member’s private schedules in a way that did not disturb the plans of the project.

7.3.4. Meeting objectives

Most of the objectives that we specified in the project specification document were met. However there were some objectives we couldn’t meet due to time constraints or decisions we had to take during development. Firstly, the subscription system is a flat rate instead of a variable cost depending on the usage of the services. This was done due to time constraints during development and would need a new cost estimation pipeline and an overall revamp of the monetization system to fix. Another objective we had in mind was to have a test-launch with a real world sample user base and test out whether our sample learned more than how much they would learn traditionally. This idea was later scrapped due to time and monetary constraints since finding such a sample would require us to arrange meetings with schools to test our product and conduct double blind experiments about how much each student learned about a certain topic. We had hoped to achieve all these objectives during development with the help of a possible sponsor we could find but development time was too scarce to spare for such a task so these ideas had to be shelved.

7.4 New Knowledge Acquired and Applied

The development of the Ubien platform required the team to learn lots of new skills and to utilize a lot of technologies to achieve our intended purpose. Building a system that orchestrates real-time client communication, asynchronous background pipelines, and interactive 3D rendering necessitated acquiring about and applying a vast array of new technologies.

A significant portion of the team's learning curve involved adopting Go for the backend infrastructure. We acquired a lot of information about Go's standard library, utilizing net/http for server architecture and html/template for server-side UI rendering. We learned to

implement lightweight routing and middleware using chi and engineered a custom, in-memory session store utilizing HTTP only cookies and bcrypt for secure password hashing. Furthermore, to handle the asynchronous nature of the course generation pipeline, the team learned to implement and manage WebSockets using the gorilla/websocket library, allowing for real-time progress and failure broadcasts to the client dashboard.

The core functionality of Ubien required the team to integrate and manage multiple cloud-based AI services. We acquired hands on experience utilizing the openai-go SDK for chat completions and educational content generation. To handle complex textbook ingestions, we integrated the Python based llama-cloud API. The team had to learn how to generate synchronized audiovisual content while utilizing Amazon Polly. We learned to extract not only Neural TTS audio but also precise viseme speech marks, which were mathematically applied to drive the lip-syncing of the 3D avatar.

On the client side, the team advanced their knowledge of vanilla JavaScript and browser APIs. We implemented XMLHttpRequest for multipart uploads with progress tracking and managed WebSocket clients for real-time UI updates. To render complex educational content, we learned to safely combine Markdown parsing, HTML sanitization, and dynamic mathematical typesetting.

One of the biggest technical challenges the team overcame was integrating a 3D environment into a standard web DOM. We acquired proficiency in Unity, utilizing the Universal Render Pipeline (URP) and glTFast for model loading. To orchestrate communication between the Go backend, the JavaScript web client, and the embedded Unity WebGL iframe, the team learned to engineer a cross-context bridge utilizing custom .jslib scripts and the browser's native postMessage API.

Finally, the team acquired extensive applied knowledge in modern DevOps practices. We utilized Docker and Docker Compose to containerize the application, ensuring parity across development and production environments. We learned to orchestrate a complex, multi-language background pipeline within these containers. This involved executing Python scripts for PDF processing alongside CLI tools like qpdf, pdftk-java, and poppler-utils. Furthermore, we incorporated a Node.js ecosystem into the pipeline container, utilizing puppeteer and @mermaid-js/mermaid-cli to render diagrams dynamically. For the development lifecycle, we integrated Air to achieve live reloading for Go, streamlining the engineering workflow.

Research about lightweight and modern tools was essential during this process and there were many iterations for all features where we could not get them to work as smoothly as we wanted. During these processes, we made heavy use of official documentations, and hands on trials to ensure what we were doing worked as we wanted.

8. Conclusion and Future Work

At present, Ubien successfully works as a standalone IBL platform. We have achieved the goal of creating a platform that utilizes Inquiry Based Learning principles with the use of LLMs to create a longer lasting learning environment. Still, there is a lot to improve and add to the system. Here are the main features we want to add to the project in the future:

- A better monetization and token usage system that allows users to know estimated course creation costs when they upload their textbook pdfs. Having a flat fee for every user is not monetarily viable and may cause overuse.

- Better custom made 3D models for instructors. Having more model types increases inclusivity and custom models can be interpreted as instructors more easily and be used as mascots for the product.
- Having smoother and more types of idle and reactionary animations in the 3D models will make the user have a better understanding of the subject and make the product feel more like a traditional personalized learning experience.
- Creating a better general textbook to course creation pipeline will allow the product to be used by more people especially since IBL isn't a very widespread way of learning and finding IBL books isn't easy.

9. Glossary

SSR: Server side rendered
API: Application Programming Interface
LLM: Large Language Model
CDN: Content Delivery Network
DOM: Document Object Model
AWS: Amazon Web Services

10. References

[1] I. Kaiser, J. Mayer, and D. Malai, “Self-Generation in the Context of Inquiry-Based Learning,” Frontiers in Psychology, vol. 9, 2018. doi: 10.3389/fpsyg.2018.02440.