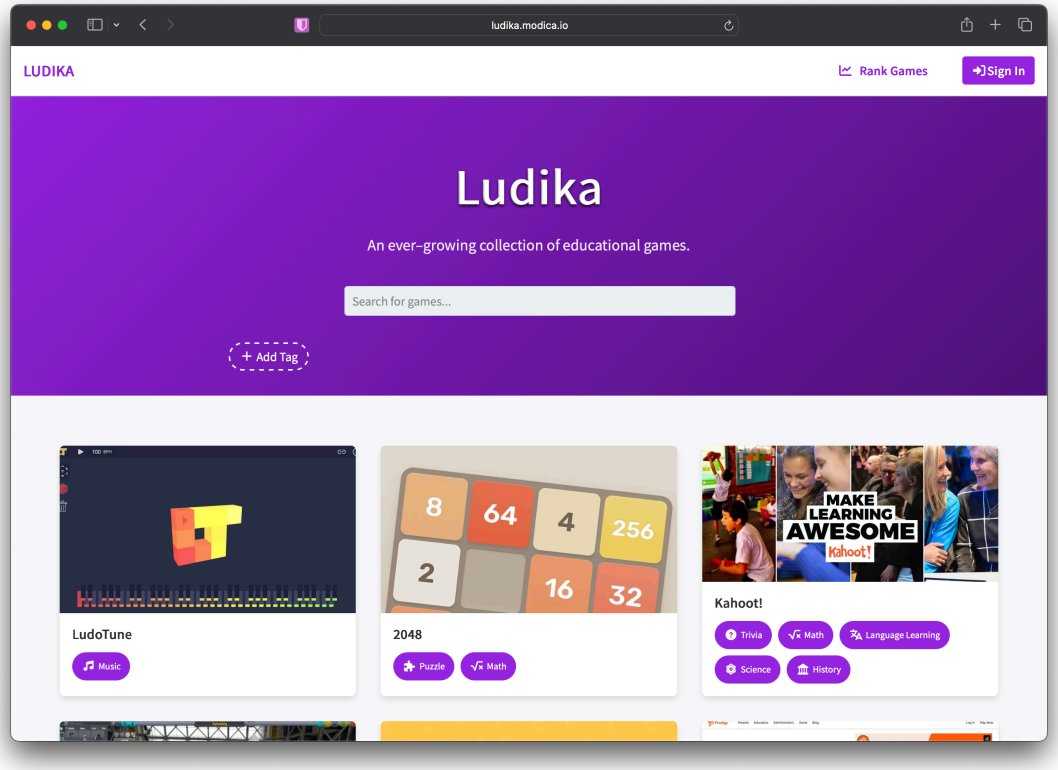
Computer Engineering Project Report

Ludika

A Platform for Game-Based Learning Tools Assessment



Made by **Alessandro Modica**
Supervising Professor **Fabrizio Amarilli**

Academic Year
2024/2025

Report made with LaTeX

POLITECNICO
MILANO 1863

Contents

1 Introduction

Game-Based Learning (GBL) is a pedagogical approach that uses games to enhance the learning experience. It has been shown to improve student engagement, motivation, and retention of knowledge. A significant market at $US 14.0 billion in 2023[1], it is projected to grow significantly in the coming years.

The growing prevalence of online learning and game-based educational tools has led many teachers and institutions to seek ways to determine the effectiveness of these tools in improving student learning outcomes. This project aims to develop a platform to enable a more systematic and data-driven approach to indexing, evaluating and discovering the best game-based learning tools.

Through a sleek user-friendly web interface, teachers and students alike can easily browse, rank, suggest and review various educational games. The platform also provides a set of metrics to help teachers gauge the effectiveness of these tools in their classrooms given their priorities. Native integration with the API of OpenAI-compatible Large Language Models (LLMs) allows for AI-facilitated categorization and addition of new games, as well as the retrieval of metadata and assets from the web and the generation of summaries, enriching the collection.

[1] Global Game-Based Learning Market Size, Share, Trends Analysis Report By Component (Solution, and Services), By Game Type (AR VR Games, AI-based Games, and Training, Knowledge & Skill-based games, language learning games, and others), By Deployment Type (On-Premises, and Cloud), By End-User, By Region and Companies - Industry Segment Outlook, Market Assessment, Competition Scenario, Trends and Forecast 2023-2032

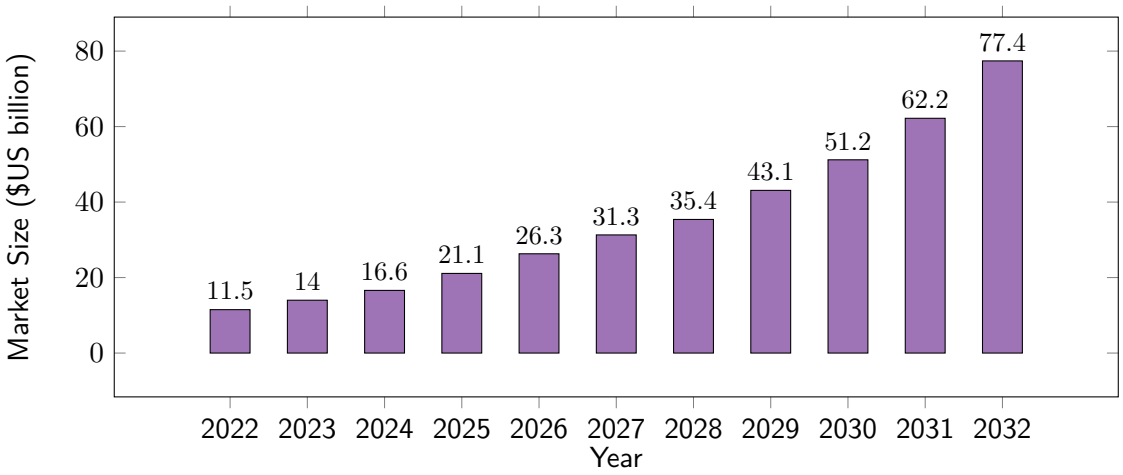


Figure 1: Growth of the Game-Based Learning Market, including the projected growth till 2032.

Given diverging priorities and needs, teachers greatly benefit from the ability to use Multi-Criteria Decision Making (MCDM) and Weighted Ranking methods to evaluate and compare educational games. The platform allows teachers to create custom evaluation profiles, assign weights to each criterion within a profile, and then rank the games based on their performance against these criteria.

In order to achieve these goals, the project was developed using a **three-tier architecture** with the API and backend logic contained within a monolithic service, in order to attain the best efficiency and reduce the footprint of service interoperation logic. The platform is built using state-of-the-art web technologies, including Nuxt for the frontend, FastAPI for the backend, and PostgreSQL for the database. LangChain provides the backbone for agentic features, while evaluations are algorithmically generated based on the games' metadata and user reviews. **Google** and **Mistral (via NVIDIA NIM)** Large Language Models (LLMs) are used to scan the web for games, and the platform of choice for scraping was **Reddit**, given the accessible and conversational nature of its content.

The use of Docker allows for easy deployment and management of the various components of the system, and the Cloudflare CDN enables fast and reliable content delivery.

2 Innovation of the proposed solution

The core value-add of Ludika is the tight integration of all the core components (game discovery, agentic web scraping, a ranking algorithm and the fine-tuned game indexing and scraping systems) into a single platform, and the bespoke agentic functionality it offers. In particular, the ability for teachers to define their own criteria makes using this platform more helpful, relevant and personal than any experience possible using a generic off-the-shelf CMS solution like Wordpress.

To achieve this, the web scraping component of the platform implements a Multi-Agent System (MAS): the go-to design for complex applications leveraging Large Language Models, it is a system composed of different specialist agents and programmatic tools arranged in a pipeline to fetch, evaluate, index, enrich with metadata and therefore categorize educational games found in organic discussions between users of the popular internet site Reddit or proposed directly by a content moderator. An additional scraper is used to fetch images and other information from the web (Wikipedia, Google and Tavily).

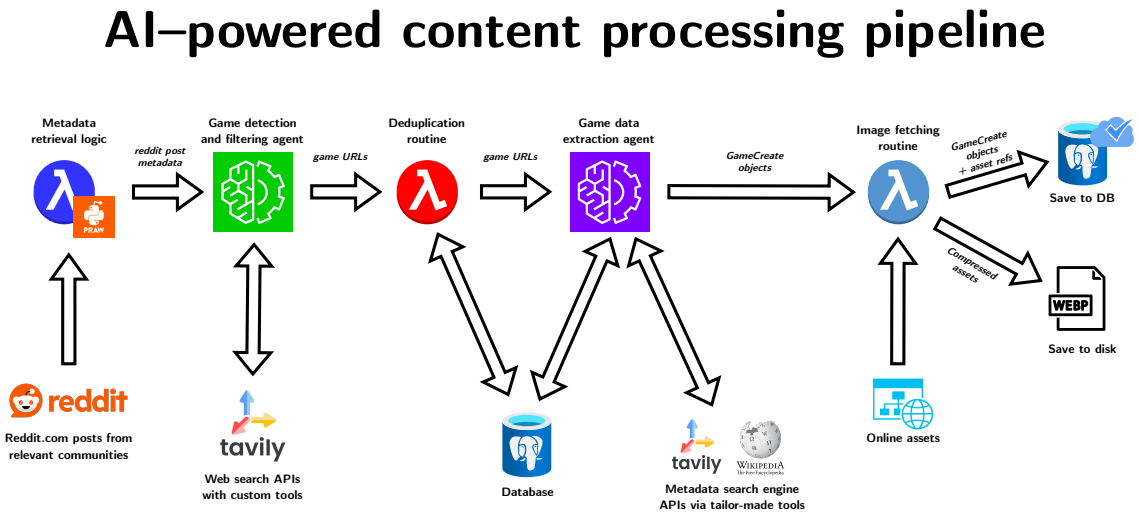


Figure 2: Agentic Web Scraping Pipeline

With the default criteria provided, teachers are able to compare their own preferences with those of their colleagues. Then, they can create their own sets of weights and see how games stack up against their specific requirements, allowing them to make more informed decisions about which games to use in their classrooms.

This platform is intended to be a living, breathing tool that evolves with the needs of its users. It has a community-driven approach to development, enabling administrators to take in suggestions and feedback from users and extend categorization and evaluation options as needed.

3 Project Development

In this section, I will provide an in-depth overview of the development process, including the rationale behind design choices, specifics about the implementation of the platform, as well as the challenges faced and lessons learned along the way.

3.1 Functionality

The platform implements a set of core functionalities that allow ***Users*** to:

- **browse and search** for educational games, either by using a search bar or by filtering through a set of predefined tags;
- **rank** educational games based on a set of criteria, including user reviews and ratings;
- **evaluate** educational games using a set of custom criteria defined by the user, allowing for a more personalized and relevant experience;
- **submit** new educational games to the platform as suggestions, allowing moderators to review and approve them;
- **review** educational games, providing feedback and ratings to help other users make informed decisions;

Additonally, ***Content Moderators*** and ***Platform Administrators*** are able to:

- **add** new educational games to the platform, either by using a web form or by using the agentic web scraping feature;
- **approve** or reject user-submitted educational games, ensuring that only high-quality and relevant content is available on the platform;
- **manage and monitor** user content and accounts, including the ability to suspend users who disrupt the platform's activities;
- **automatically scrape** Reddit communities for new educational games, using a set of predefined rules and heuristics to identify relevant posts and fetch additional metadata from the web;

3.2 Technology Stack

The platform is built upon a robust technology stack that includes:

- **Frontend:** Nuxt, a powerful framework for building responsive server-rendered applications with Vue, providing a seamless user experience and fast performance. Presentation layer processing is thus partly offloaded to the server, allowing for faster page loads and improved SEO. Nuxt is built on top of Vue, a popular JavaScript framework for building user interfaces, and provides a powerful set of features for building applications for the modern web.
- **Backend:** FastAPI, a modern web framework for building APIs with Python based on standard Python type hints, first-class citizen support for Pydantic models and an asynchronous runtime conforming to the ASGI standard, ensuring high performance and easy development.
- **Database:** PostgreSQL, a powerful open-source relational database system that provides reliability and data integrity. Its advanced features, such as support for JSON object storage, full-text search and concurrency synchronization semantics via *advisory locks*, make it an ideal choice for this project.
- **Containerization:** Docker, allowing for easy deployment and management of the various components of the system. Docker implements the OCI runtime specification, which allows for the use of OCI-compliant container images and runtimes in any environment that supports the OCI standard, such as Kubernetes, OpenShift, Amazon ECS, and Azure Container Instances.
- **ORM:** SQLModel, a Python library that provides a simple and efficient way to interact with databases using Python objects. It is built on top of SQLAlchemy and Pydantic, providing a powerful and flexible ORM solution for FastAPI applications reducing code repetition and improving maintainability.
- **Web Scraping:** Beautiful Soup, a Python library for web scraping that allows for easy extraction of data from HTML and XML documents. It provides a powerful and flexible framework for building web crawlers allowing users to add new games with minimal human intervention.
- **Reddit Integration:** PRAW, the Python Reddit API Wrapper, which provides a powerful interface for interacting with the Reddit API. It allows for easy integration with Reddit, enabling the platform to fetch and index new educational games from Reddit communities.
- **CDN:** Cloudflare, providing fast and reliable content delivery through its global network.
- **LLM Integration:** LangChain, enabling the use of compatible Large Language Models for automated categorization and information retrieval.

3.3 Architecture

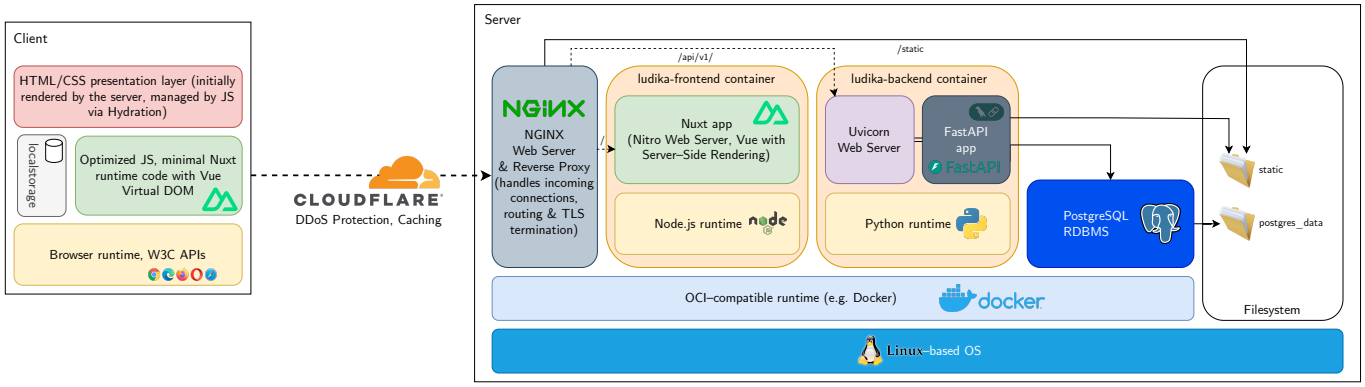


Figure 3: Architectural Diagram

The architecture of the platform is designed to be modular and horizontally scalable. **Most of the functionality is delegated to the server, while the client is exclusively responsible for the presentation layer, provided by Nuxt and also partially rendered on the server**.

The system is composed of several key components, each designed to be independently deployable and scalable, allowing for efficient resource utilization and easy maintenance. In the active deployment, all components are hosted on a single node, but it is possible to scale the API server, database and Nuxt SSR independently, thanks to the stateless nature of the API and the use of a relational database as the single source of truth for state. The main components are:

- **FastAPI Backend**: the core of this project, providing a RESTful API to interact with entities. Also handles authentication and authorization, and executes Agentic routines to scrape and index new educational games from the web.
- **PostgreSQL Database**: the single record information store for the platform, storing all data except for static assets served directly by NGINX.
- **Nuxt Frontend**: a single-page application that provides a user-friendly interface for interacting with the platform.
- **NGINX**: a web server that serves static assets and acts as a reverse proxy for the API and Nuxt applications.

All of these services can be deployed quickly using **Docker Compose**, which allows for easy management of the various components and their dependencies. The platform is designed to be easily deployable on any cloud provider or on-premises infrastructure.

In addition, the app depends on the following third-party services:

- **Cloudflare**: a CDN that provides fast and reliable content delivery, as well as security features such as DDoS protection and SSL termination.
- **Reddit API**: used to fetch new educational games from Reddit communities, allowing the platform to stay up-to-date with the latest trends in game-based learning.
- **NVIDIA NIM API**: used to access the Large Language Model (LLM) for agentic web scraping and categorization of educational games.

- **Tavily and Wikipedia APIs**: used to programmatically search the web.
- **Google CSE API**: used to retrieve images for educational games.

3.3.1 Data Model

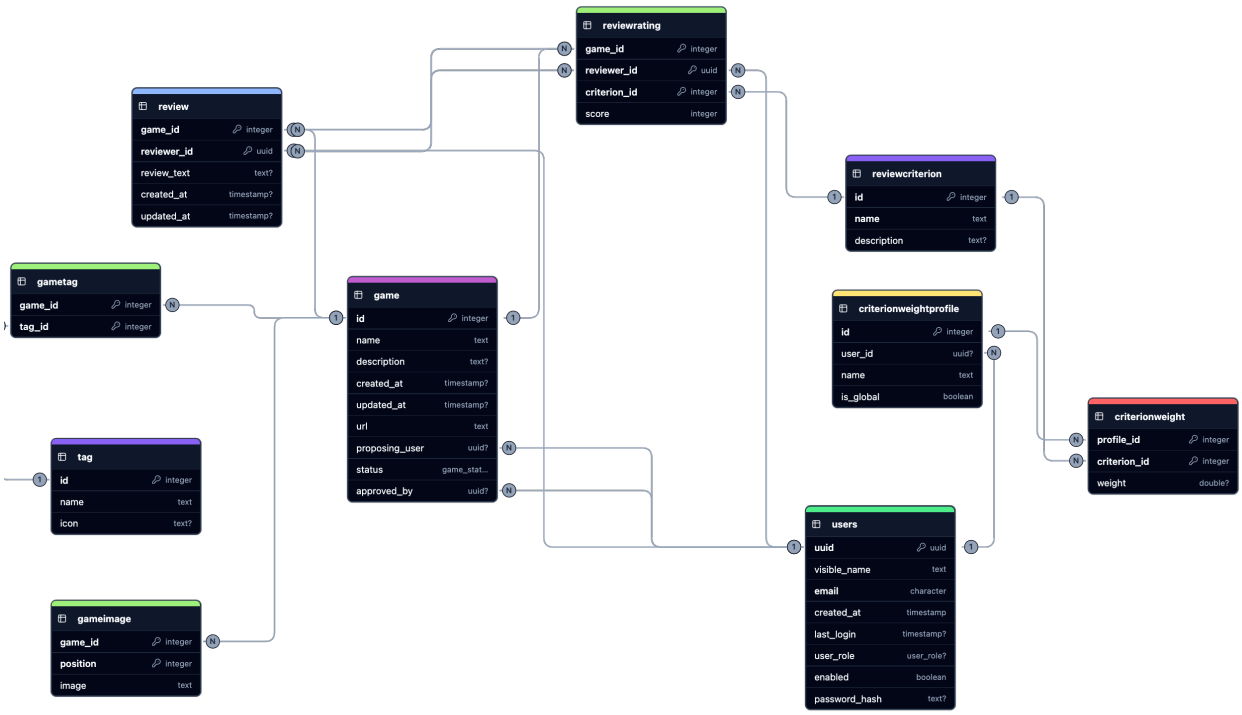


Figure 4: Data Model

The data model is designed with efficiency and simplicity in mind, using SQL–native constraints and indexes to ensure maximum performance. The following entities and relations are defined:

- **Users**: Registered accounts, each with a unique UUID, visible name, email, creation and last login timestamps, a user role (`user`, `content_moderator`, or `platform_administrator`), enabled status, and password hash. Disabled users are prevented from logging in.
- **Game**: Educational games proposed by users, with fields for name, description, creation and update timestamps, URL, proposing user, status (`draft`, `submitted`, `approved`, or `rejected`), and the approving user.
- **GameImage**: Images associated with each game, identified by game ID and position, storing the image path or reference.
- **Tag**: Tags that can be assigned to games, each with an ID, name, and optional icon. Tags are used to categorize games and allow for easy filtering and searching.
- **GameTag**: Association table linking games and tags (many-to-many relationship).
- **ReviewCriterion**: Criteria used for reviewing games (e.g., “Ease of use”, “Fun factor”), each with a unique name and description.

- **Review**: Reviews of games by users, including review text, creation and update timestamps, and identified by game and reviewer.
- **ReviewRating**: Ratings for each criterion in a review, storing the score (1–5) for each criterion, game, and reviewer.
- **CriterionWeightProfile**: Weight profiles defined by users (or admins if global), with a name and a flag indicating if the profile is global.
- **CriterionWeight**: Weights for each criterion in a weight profile, associating a profile with a criterion and a non-negative weight.

The project also makes use of the following custom PostgreSQL enum types:

- `user_role`: {`user`, `content_moderator`, `platform_administrator`}
- `game_status`: {`draft`, `submitted`, `approved`, `rejected`}

The **class diagram** (UML) of the Python data model matches the structure of the database schema, thanks to the use of SQLModel, fully deduplicating model definition across SQL and Python code.

3.3.2 API Model

The API, built using FastAPI, uses a RESTful architecture with JSON responses, Bearer authentication and OpenAPI documentation. The following endpoints are available:

- `/`:

GET / — Status

Description: Root endpoint that returns status information.
Response: `application/json (json)`

- `/status`:

GET /status — Status

Description: Root endpoint that returns status information.
Response: `application/json (json)`

- `/games/`:

GET /games/ — Get Games

Description: Retrieve a list of all approved games with pagination, tag filtering and search.
Response: `application/json (json)`

POST /games/ — Create Game

Description: Create a new game.
Request Body: `application/json`
Response: `application/json (json)`

- `/games/my-games`:

GET /games/my-games — Get My Games

Description: Get games created by the current user.
Response: `application/json (json)`

- `/games/waiting-for-approval`:

GET /games/waiting-for-approval — Get Games Waiting For Approval

Description: Get games waiting for approval (privileged users only).
Response: `application/json (json)`

- `/games/{game_id}`:

GET /games/{game_id} — Get Game

Description: Retrieve a game by its ID.
Response: `application/json (json)`

DELETE /games/{game_id} — Delete Game

Description: Delete a game (only by creator or privileged users).
Response: `application/json (json)`

PATCH /games/{game_id} — Update Game

Description: Update a game (only by creator or privileged users).
Request Body: `application/json`
Response: `application/json (json)`

- `/games/{game_id}/with-reviews`:

GET /games/{game_id}/with-reviews — Get Game With Reviews

Description: Retrieve a game by its ID with reviews included.
Response: `application/json (json)`

- `/games/{game_id}/images/{image_no}`:

GET /games/{game_id}/images/{image_no} — Get Game Image

Description: Get a specific image for a game.
Response: `application/json (json)`

PUT /games/{game_id}/images/{image_no} — Replace Game Image

Description: Replace an existing image for a game.
Request Body: `multipart/form-data`
Response: `application/json (json)`

DELETE /games/{game_id}/images/{image_no} — Delete Game Image

Description: Delete a specific image from a game.
Response: `application/json (json)`

- `/games/{game_id}/images`:

POST /games/{game_id}/images — Post Game Image

Description: Upload a new image for a game.
Request Body: `multipart/form-data`
Response: `application/json (json)`

- `/games/ranked/{profile_id}`:

GET /games/ranked/{profile_id} — Get Ranked Games

Description: Get ranked games for a given profile.
Response: `application/json (json)`

- `/tags/`:

GET /tags/ — Get Tags

Description: Get all tags.
Response: `application/json (json)`

POST /tags/ — Add Tag

Description: Create a new tag (admin only).
Request Body: `application/json`
Response: `application/json (json)`

- `/tags/{tag_id}`:

DELETE /tags/{tag_id} — Delete Tag

Description: Delete a tag (admin only).
Response: `application/json (json)`

PATCH /tags/{tag_id} — Update Tag

Description: Update a tag (admin only).
Request Body: `application/json`
Response: `application/json (json)`

- `/users/`:

GET /users/ — List Users

Description: Get all users (privileged users only).
Response: `application/json (json)`

- `/users/me`:

GET /users/me — Get Me

Description: Get the current user.
Response: `application/json (json)`

- `/users/me/visible-name`:

PATCH /users/me/visible-name — Update Visible Name

Description: Update the current user's visible name.
Request Body: `application/json`
Response: `application/json (json)`

- `/users/me/password`:

PATCH /users/me/password — Update Password

Description: Update the current user's password.
Request Body: `application/json`
Response: `application/json (json)`

- `/users/{user_id}`:

GET /users/{user_id} — Get User

Description: Get a specific user by ID (privileged users only).
Response: `application/json (json)`

PATCH /users/{user_id} — Admin Update User

Description: Update a user (privileged users only).
Request Body: `application/json`
Response: `application/json (json)`

DELETE /users/{user_id} — Delete User

Description: Delete a user (admin only).
Response: `application/json (json)`

- `/users/{user_id}/games`:

DELETE /users/{user_id}/games — Delete User Games

Description: Delete all games created by a user (moderators and admins only).
Response: `application/json (json)`

- `/reviews/criteria`:

GET /reviews/criteria — List Criteria

Description: Get all review criteria.
Response: `application/json (json)`

POST /reviews/criteria — Create Criterion

Description: Create a new review criterion (admin only).
Request Body: `application/json`
Response: `application/json (json)`

- `/reviews/criteria/{criterion_id}`:

PATCH /reviews/criteria/{criterion_id} — Update Criterion

Description: Update a review criterion (admin only).
Request Body: `application/json`
Response: `application/json (json)`

DELETE /reviews/criteria/{criterion_id} — Delete Criterion

Description: Delete a review criterion (admin only).
Response: `application/json (json)`

- `/reviews/profiles`:

GET /reviews/profiles — List Profiles

Description: Get available criterion weight profiles.
Response: `application/json (json)`

POST /reviews/profiles — Create Profile

Description: Create a new criterion weight profile.
Request Body: `application/json`
Response: `application/json (json)`

- `/reviews/profiles/{profile_id}`:

GET /reviews/profiles/{profile_id} — Get Profile

Description: Get a criterion weight profile.
Response: `application/json (json)`

PATCH /reviews/profiles/{profile_id} — Update Profile

Description: Update a criterion weight profile.
Request Body: `application/json`
Response: `application/json (json)`

DELETE /reviews/profiles/{profile_id} — Delete Profile

Description: Delete a criterion weight profile.
Response: `application/json (json)`

- `/reviews/{game_id}`:

GET /reviews/{game_id} — Get Game Reviews

Description: Get all reviews for a specific game.
Response: `application/json (json)`

- `/reviews/{game_id}/my-review`:

GET /reviews/{game_id}/my-review — Get My Review

Description: Get the current user's review for a specific game, if it exists.
Response: `application/json (json)`

PUT /reviews/{game_id}/my-review — Create Or Update My Review

Description: Create a new review or replace existing review for the current user. Only allowed for approved games.
Request Body: `application/json`
Response: `application/json (json)`

DELETE /reviews/{game_id}/my-review — Delete My Review

Description: Delete the current user's review for a specific game, if it exists.
Response: `application/json (json)`

- `/reviews/{game_id}/{user_id}`:

GET /reviews/{game_id}/{user_id} — Get User Review

Description: Get a specific user's review for a specific game, if it exists.
Response: `application/json (json)`

DELETE /reviews/{game_id}/{user_id} — Delete User Review

Description: Delete a specific user's review for a specific game. Only privileged users can do this.
Response: `application/json (json)`

- `/auth/signup`:

POST /auth/signup — Signup

Description: Register a new user account.
Request Body: `application/x-www-form-urlencoded`
Response: `application/json (json)`

- `/auth/login`:

POST /auth/login — Login

Description: Authenticate a user and return an access token.
Request Body: `application/x-www-form-urlencoded`
Response: `application/json (json)`

- `/ai/add-game-from-url`:

POST /ai/add-game-from-url — Create Game From Url

Description: Create and immediately approve a new game given a valid URL, with the power of AI!
Response: `application/json (json)`

- `/ai/generate-game-from-url`:

POST /ai/generate-game-from-url — Generate Game Object From Url

Description: Generate a GameCreate object from a URL without saving it to the database. This endpoint uses AI to analyze the web page and create a structured game object.
Response: `application/json (json)`

- `/ai/reddit-scraping`:

GET /ai/reddit-scraping — Get Reddit Scraping Session

Description: Get the current Reddit scraping status.
Response: `application/json (json)`

POST /ai/reddit-scraping — Create Reddit Scraping Session

Description: Process Reddit posts to find educational games and generate game objects automatically.
Response: `application/json (json)`

- `/ai/test-reddit-fetch`:

GET /ai/test-reddit-fetch — Test Reddit Loader

Response: `application/json (json)`

3.3.3 Authentication & Authorization

The authentication system uses JWT, a widely adopted standard that allows for stateless and lightweight authentication without compromising security. OAuth2 bearer tokens are used to authenticate requests, and passwords are securely hashed using the `Argon2` algorithm, an industry standard.

Authorization is arranged around a simple Role-Based Access Control (RBAC) scheme, with three roles: `user`, `content_moderator` and `platform_administrator`. Each role has a different set of permissions, allowing for control over what registered users can do on the platform.

On the client side, an authenticated wrapper of the `fetch` web API, `authenticatedFetch`, is used to automatically validate the user's authentication state and access gated endpoits. This wrapper is used throughout the frontend to ensure that all requests are authenticated and that the user has the necessary permissions to access the requested resources.

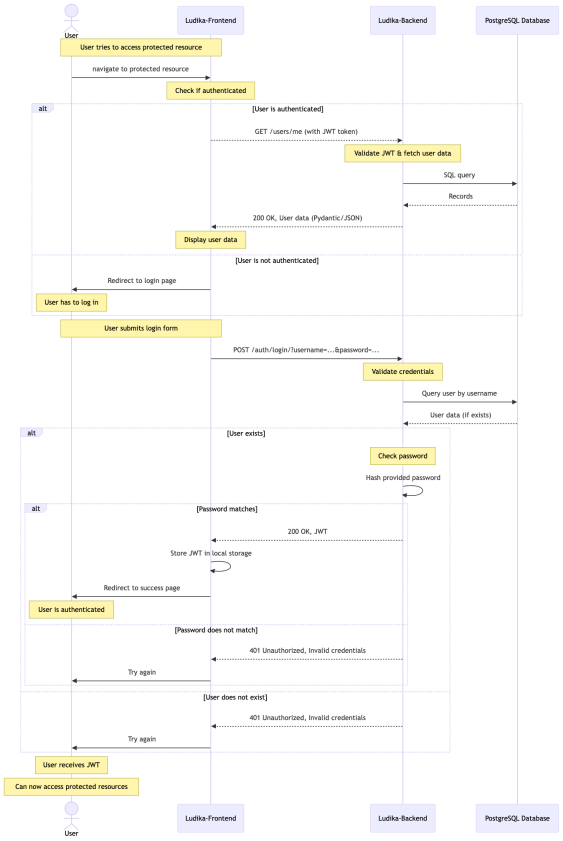


Figure 5: Login Sequence Diagram

3.3.4 Frontend

The front-end's design reflects on the core principles of Vue, Nuxt and reactive web development: the use of Vue *Composables* to abstract away data and state management from views, the implementation of several reusable and parametrized *Com-*

ponents—built upon the many fragments available in the Vuestic UI component library—that appear within pages, and the use of native Nuxt routing mechanisms to organize the structure of the website.

The result is a pleasing–to–use, responsive and modern–looking single-page application, with snappy transitions between routes that feel much more akin to a native experience than most traditional web-pages.

3.4 Development Process

Development of the project was carried out in a linear fashion, with a set of milestones that were completed in the following order.

3.4.1 API foundation

The API was developed first, using PostgreSQL, FastAPI and SQLModel to define the data models and the API endpoints. The goal was to have a solid foundation for the backend, and to be able to test the API using the interactive OpenAPI documentation. Authentication and authorization were implemented using JWT tokens and OAuth2 bearer authentication. Access control is implemented in the form of a simple RBAC model and with salted and hashed passwords. Swagger UI allows for easy testing of the API endpoints with examples, an OAuth login form, and docstrings in the code are automatically included in the OpenAPI documentation, making it easy to understand the purpose and usage of each endpoint. Additionally, a simple bespoke Python script was developed to convert the OpenAPI documentation to LaTeX code, for inclusion in this report.

Here's an example of a protected endpoint:

```python
@game_router.post("/")
async def create_game(
    game: GameCreate,
    db_session: Session = Depends(get_session),
    current_user: User = Security(get_current_user),
) -> GamePublic:
    """Create a new game."""
    ...
```

The request dependency features of FastAPI make it easy to ensure a consistent and secure flow when accessing an Access Control–enforced endpoint, or one that depends on external resources (e.g. the database). Similarly, Pydantic models and semantic type hints in FastAPI make it easy to ensure that the request body is validated and that no more data than necessary is returned to the client.

Another security dependency, `Security(get_current_user_optional)` was implemented to allow for optional authentication:

```python
@game_router.get("/{game_id}")
async def get_game(
    game_id: int,
    db_session: Session = Depends(get_session),
    current_user: User | None = Security(get_current_user_optional),
) -> GamePublic:
    """Retrieve a game by its ID."""
    statement = select(Game).where(Game.id == game_id)
    if current_user:
        if not current_user.is_privileged:
            statement = statement.where(
                or_(
```

```
                    Game.proposing_user == current_user.uuid,
                    Game.status == GameStatus.APPROVED.value,
                )
            )
    else:
        statement = statement.where(Game.status == GameStatus.APPROVED.value)
    ...
```

Optional authentication is useful for endpoints that are not protected by default but enable additional functionality for users who are authenticated, such as accessing a game that was not approved yet but was proposed by the user themselves. This makes the API more simple to use and more robustly conformant to RESTful design principles, eliminating the need for separate endpoints for authenticated and unauthenticated requests.

3.4.2 Frontend

The frontend was developed using Nuxt, a Node framework for building server-rendered applications with Vue. The goal was to create a responsive, mobile-friendly single-page application that exposes all the functionality implemented in the API.

To that end, open-source components from the Vuestic UI library were used in conjunction with custom components to create a modern and visually appealing UI. Nuxt Images is used for optimized loading of images, and a variety of custom composables handle the interaction with the API.

When the user opens the website, they are immediately presented with a store-like view of the available games, with a search bar and a tag selector to fine-tune the results.

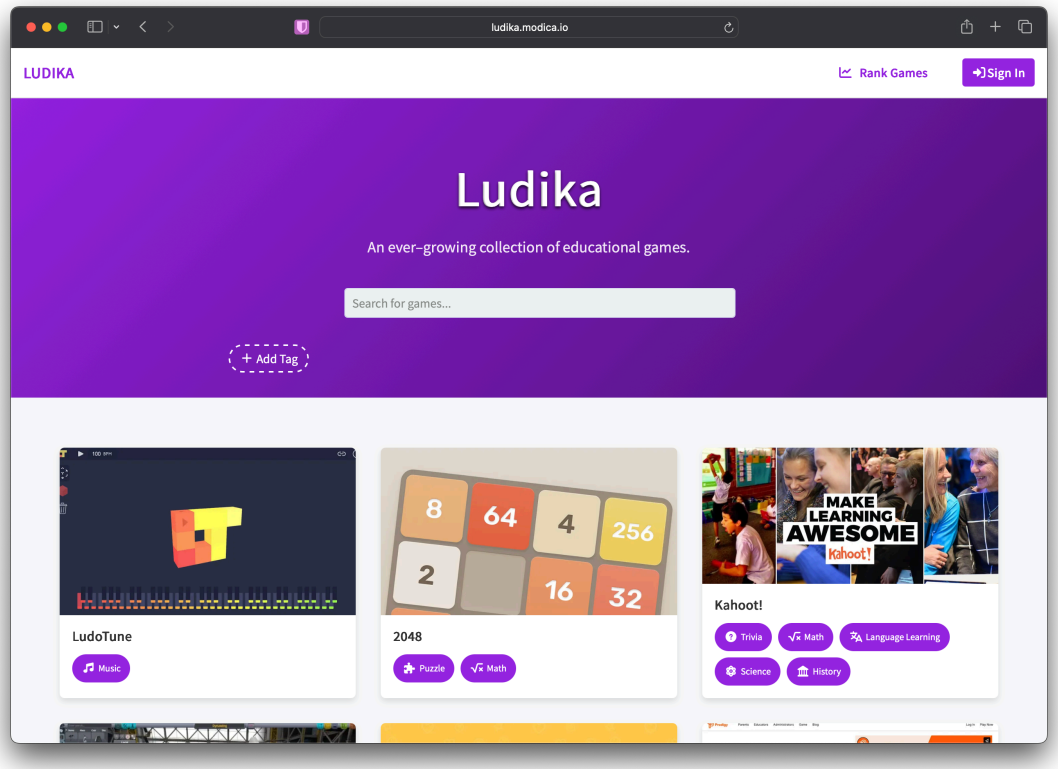


Figure 6: Ludika Homepage

Much care went into designing the game detail page, which provides a comprehensive overview of

the game along with its metadata and image slideshow, but is also home to the game's reviews and ratings:

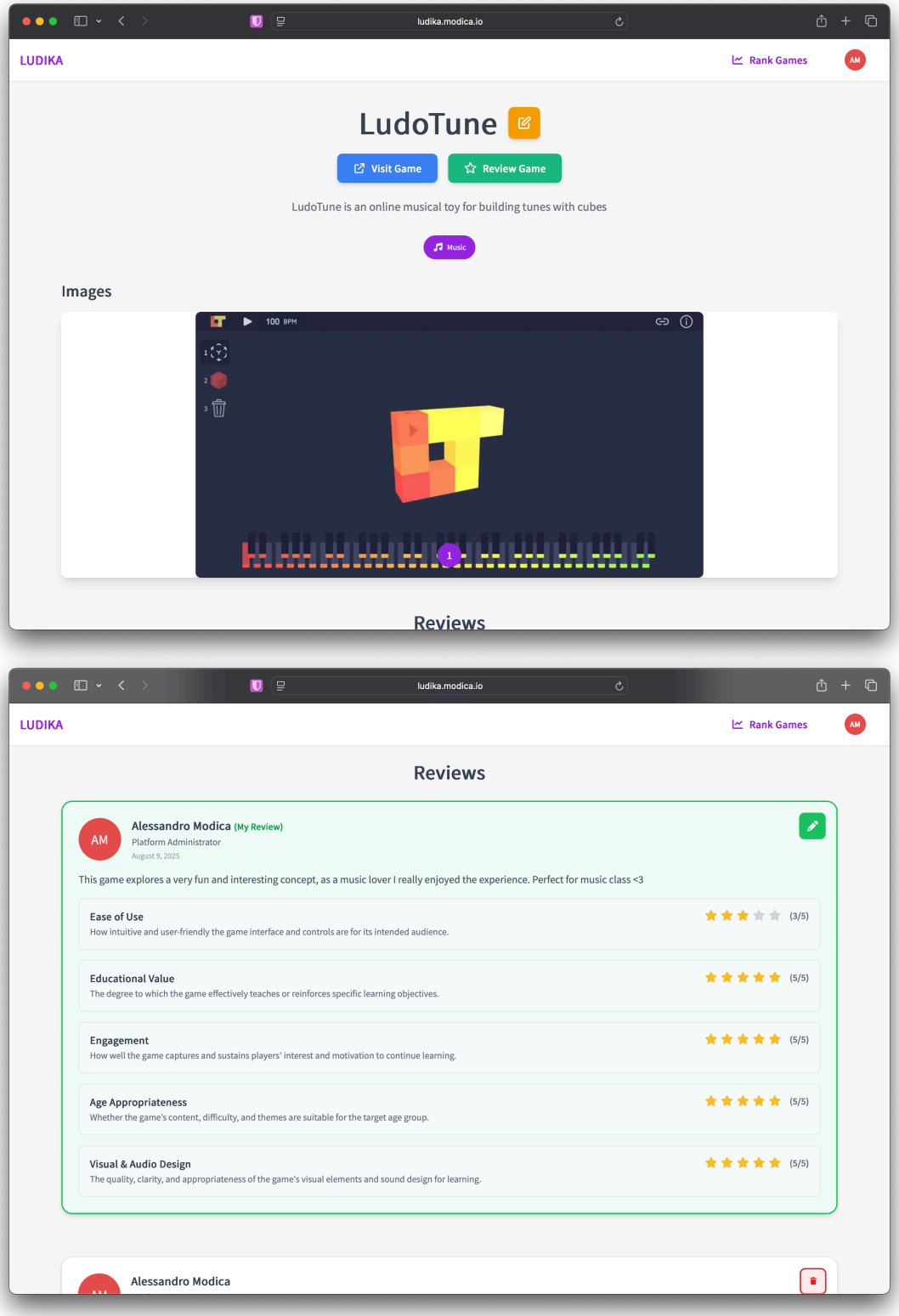


Figure 7: Ludika Game Detail Page, showing a user's review

Users must log in to leave a review. Anyone can freely create an account through the registration page, and users get logged in automatically after registration.

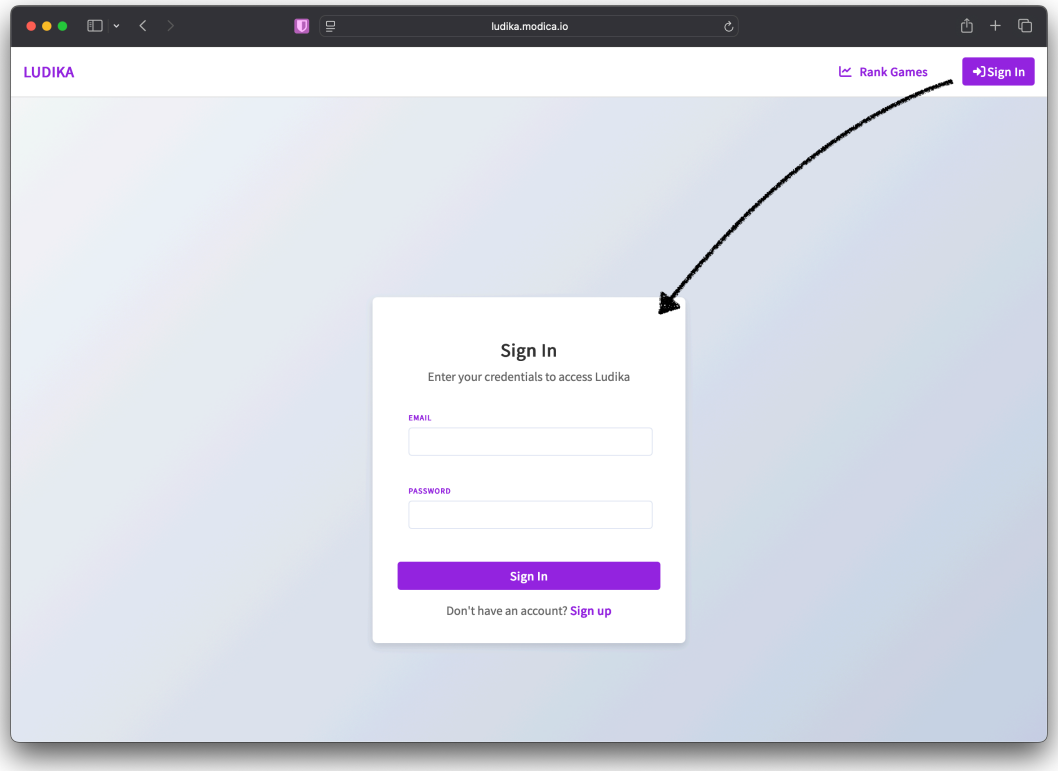


Figure 8: The sign-in page, which can be accessed from anywhere and automatically opens when the user tries to access a protected route.

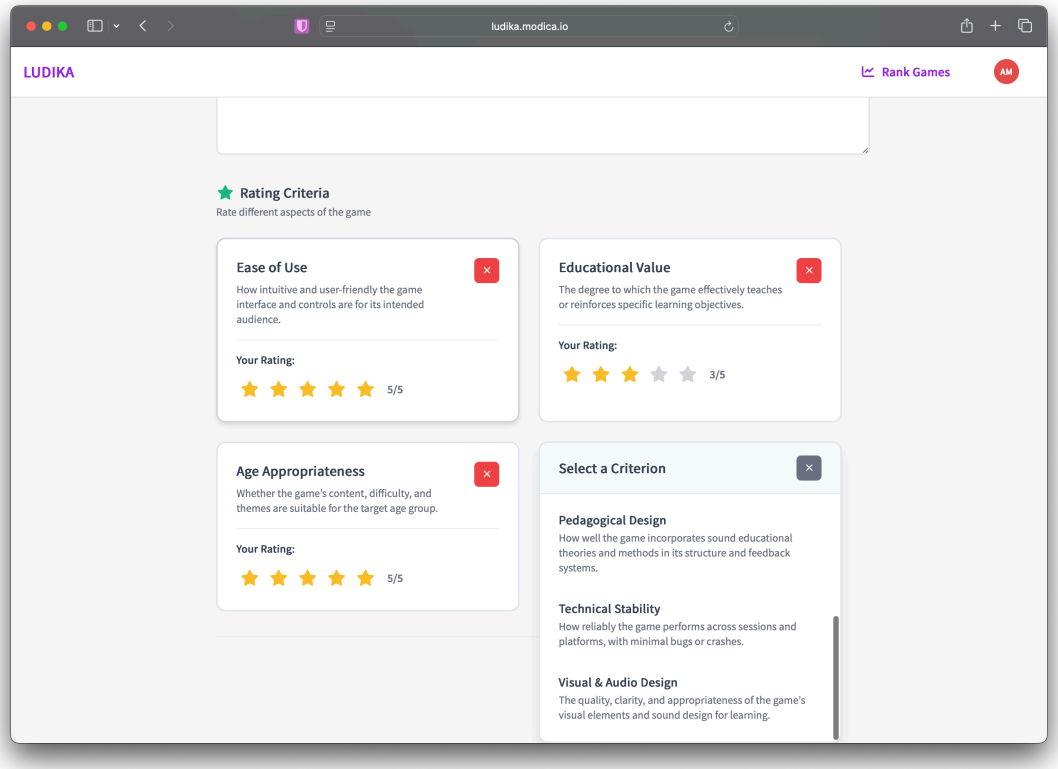


Figure 9: Users can review games to provide feedback and ratings. Each review can include any number of ratings from the available criteria.

The core feature of this project—comparative game evaluation—is implemented in the *Rank Games* section, accessible from the top bar.

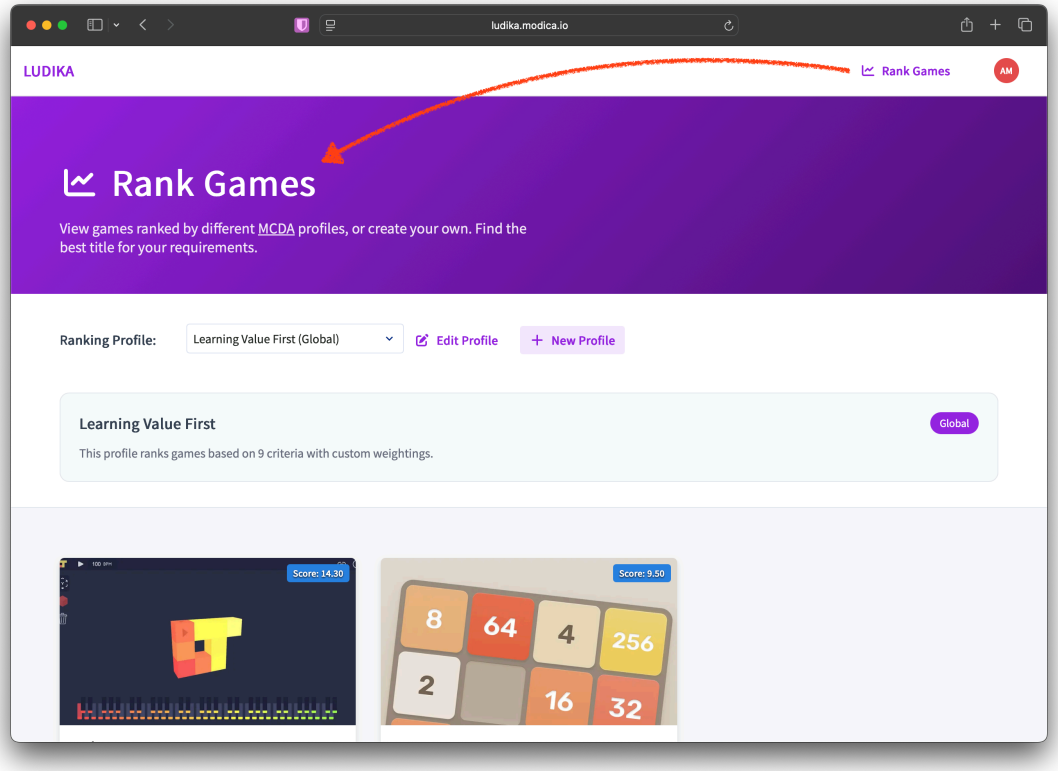


Figure 10: *Rank Games* shares its `GameLibrary` component with the main screen, as well as other views (e.g. the *My Games* screen), with `prop`-based customization options to adapt it to the specific use case.

Vue props are used extensively throughout the application to pass configuration and data assignments to custom components. This makes it as easy as this to define a new view using the `GameLibrary` component:

```
<template>
    <div>
        <GameLibrary title="My Games"
                     subtitle="Manage your created games"
                     apiEndpoint="/api/v1/games/my-games"
                    :showHero="true" :searchable="true"
                    :tagFilterable="false"
        />
    </div>
</template>
```

Forms are also widely used within the application. Some forms also integrate cards and choice dropdowns to provide a more interactive and intuitive editing of structured items. Such is the case for reviews (as seen above), as well as the editing of custom ranking profiles for games.

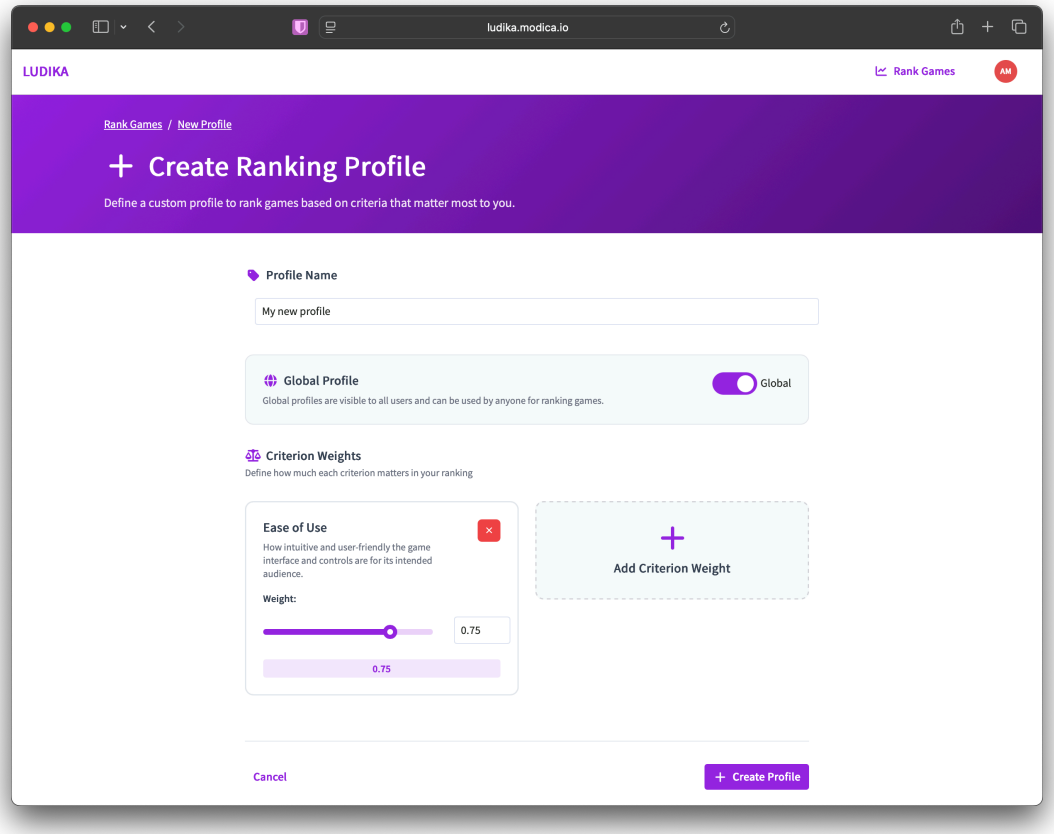


Figure 11: The ranking profile editor allows users to create custom profiles with a set of criteria and weights, which can then be used to rank games.

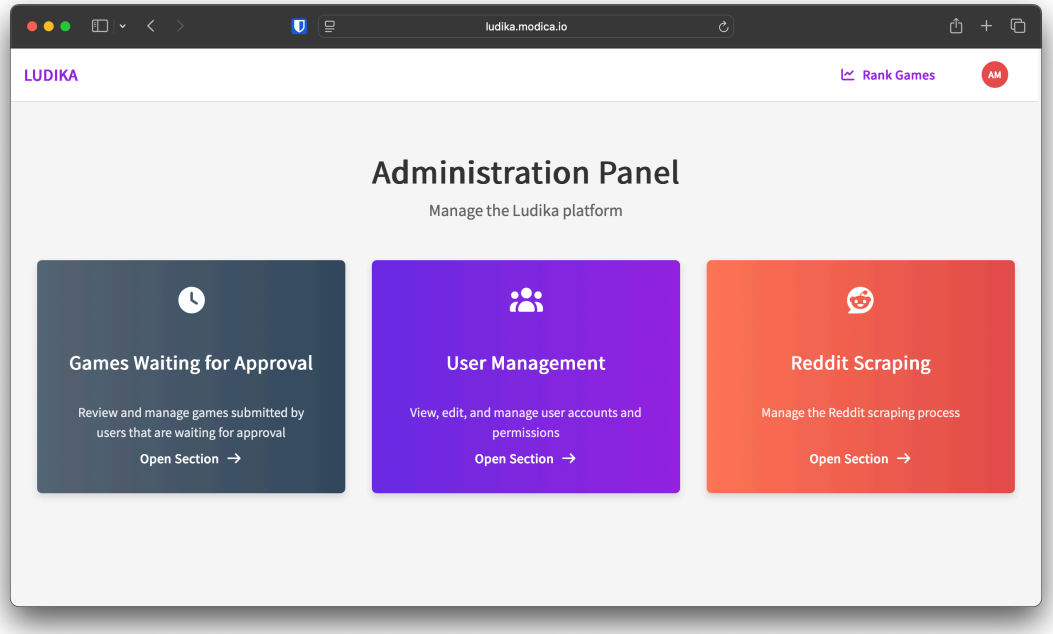


Figure 12: The admin panel provides a comprehensive overview of the system, allowing privileged users to manage users, games, and the scraping facilities.

3.4.3 Agentic Web Scraping

Agentic Web Scraping is at the heart of two features in Ludika: the **automated, background scraping of social media/forum communities** (Reddit) for new educational games to add to the platform, and **the fetching of game metadata and assets from the web** to speed up the onboarding process for new games proposed by content moderators.

- **Reddit scraping** is implemented using the `RunnableLambda` paradigm of LangChain, which allows for parallel processing of tasks. In the first stage, posts are fetched from Reddit communities via the site's JSON API, using the PRAW library to programmatically talk to Reddit.

  Then, the posts are processed and filtered to extract metadata, which is fed into the LangChain pipeline for detection of relevant game URLs. This first **Detection Agent** has access to a `TavilySearch` tool that uses the Tavily API to gain insights from the web, if needed.

  A simple check is run against the URL to ensure it's not already present in the database, and then URLs are fed into the subsequent stage of the pipeline, where a second **Metadata Retrieval Agent** finds additional information using the Tavily API, Wikipedia (via a custom `wikipedia_search` tool), as well as metadata from the URL's webpage itself.

  This second agent also has access to tools to interact with the database and in particular also selects `Tags` to be associated with the new game, as well as checks if the game is present in the DB, in case the previous check did not catch this duplication. A structured response is output from the agent with a special tool, `create_game_with_fixed_url`; this is a parametrized tool that is instantiated when setting up the agent's pipeline, ensuring that AI "hallucination" can at least not interfere with the URL provided by the site (or user).

  One final step leverages the power of Google's CSE (Custom Search Engine) API to find images for the game. These are fetched using *Requests*

- **Metadata retrieval** for user-submitted games essentially uses only the second stage of the pipeline described above, where the user-provided URL is passed to the `Metadata Retrieval Agent` to fetch additional information about the game. This includes the game's name, description, tags, and images. The agent uses the same tools as before, and the ID of the resulting game is returned to the user, who can immediately view the result of the agentic generation.

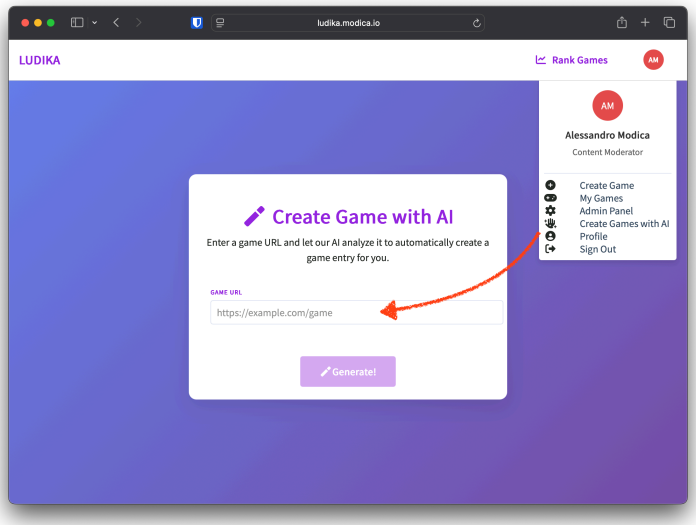


Figure 13: The UI for creating a new game with AI assistance in Ludika.

4 Results & Conclusion

The final design attempts to blend user-friendly interfaces with effective API design and a wholly robust and maintainable package. The user interface is designed to work on both desktop and mobile devices with any screen size, and the API is built to be easy to extend and to have minimum attack surface in terms of security concerns.

Development took place over the course of a few months in the 2nd semester of Academic Year 2024/2025, with the first few weeks dedicated to researching technology, planning out the architecture and data model design, reading documentation and experimenting with the various components of the stack.

4.1 Challenges with the implementation & original value additions

Several challenges were encountered during the implementation of the Ludika platform. One of the primary aims was ensuring seamless integration between the various components of the stack, particularly the interaction between the FastAPI backend and the Nuxt frontend. This required careful planning and design of the endpoints, the establishment and following of several conventions for data formats and methods (visible in the official API Documentation), and the creation of useful wrappers on the client–side to minimize the chances of diverging behaviors in different parts of the app.

Another challenging aspect was the implementation of the agentic feature set, which involved creating a system that could intelligently respond to input data and make decisions based on the context of operation. This required extensive testing and tweaking of the prompts and representation formats used to talk with the LLMs, as well as fine-tuning the detection algorithms and tooling to reduce unintended behavior. One key aspect of this was parametrizing and delegating as much of the orchestration/decision–making process as possible to deterministic logic. For instance, the initialization of the agent pipeline takes advantage of Python's functional programming elegance and thread synchronization features to create a pipeline with forward–passing of context and safe concurrent access to observability metrics:

```
def pipeline_factory(self):
    """Create a pipeline for processing Reddit posts"""
    def detect_and_process(post: RedditPost):
        raw_detection = game_detection_executor.invoke(
            {"post_content": f"{post.title}"
                             f"{'\n\n' + post.url if post.url else ''}\n\n"
                             f"{post.selftext}"
            }
        )

        detection = DetectionResult(**json.loads(raw_detection["output"]))
        self._posts_processed_counter.increment()

        if detection.has_game_url and detection.url:
            self._games_found_counter.increment()
            get_logger().info(f"Found game URL: {detection.url}"
                              f" for post: {post.title}")
            with db_context() as db_session:
                if db_session.exec(
                    select(Game)
                    .where(Game.url == detection.url)
                ).first():
                    return {
```

```
                        "skipped": True,
                        "reason": "Game already exists",
                        "post_title": post.title
                    }
            executor = create_agent_executor_for_game_create(
                detection.url,
                game_added_callback=lambda:
                    self._games_added_counter.increment()
            )
            return executor.invoke({})

        return {
            "skipped": True,
            "reason": "No game URL detected",
            "post_title": post.title
        }

    def complete_pipeline():
        self._posts = get_top_posts()
        self._posts_found = len(self._posts)
        RunnableLambda(detect_and_process).batch(self._posts)

    return complete_pipeline
```

The final implementation of the AI controller, while not entirely free from issues stemming from LLM non–determinism (which require some degree of human oversight to vet data as it is imported), is robust enough to have been used to populate the majority of the entries on the platform.

The project has the potential to significantly enhance the user experience for teachers and students by providing a more interactive and engaging platform for discovering and evaluating educational games than existing websites within its niche. The community–based approach to growing its library and the use of AI to streamline the game creation process is a key differentiator for Ludika, allowing the platform to grow organically while also gaining insight into new titles within the Game–Based Learning (GBL) space from online activity with no direct human intervention.

4.1.1 Demo & Links

- ludika.modica.io: the live platform
- Live API Documentation: the OpenAPI documentation for the API, which can be used to interact with the platform programmatically.
- TheManchineel/ludika: the source code for the project, available on GitHub.

5 Bibliography

5.1 Reference Documentation

- PostgreSQL Official Documentation: reference for the database system, including schema configuration, data types and advanced query features.
- Nuxt Documentation: reference for the frontend framework, including routing, state management and server-side rendering.
- FastAPI Documentation: used as reference for building the API, developing authentication and authorization, and implementing the database models.
- LangChain Documentation: reference for the integration of OpenAI-compatible Large Language Models, including the use of agents, callables, chains and document retrievers.
- BeautifulSoup Documentation: reference for the web document parsing library, used to fetch and index new educational games.
- SQLModel Documentation: reference for the ORM library, used to interact with the database and define the data models.

5.2 Articles & Whitepapers

- An Introductory Guide to Multi-Criteria Decision Analysis (MCDA): a comprehensive overview of the MCDA methodology, including its principles, applications and limitations.
- Accurate multi-criteria decision making methodology for recommending machine learning algorithm: a research paper that discusses the use of MCDA for recommending machine learning algorithms, providing insights into the methodology and its applications.

5.3 Other Projects and Resources

- Nuxt.js FastAPI Starter: a boilerplate project that provides a starting point for building applications using Nuxt.js and FastAPI.
- Building a simple e-commerce with Nuxt.js and FastAPI – Azerbaijan Python User Group: a series of articles that involve the creation of a project similar in scale and complexity to this one, providing a good reference for the development process of a Nuxt.js/FastAPI application.