KING'S College LONDON

Faculty of Natural, Mathematical, and Engineering Sciences

Department of Informatics

Software Engineering Group Project

ReWork

Pixel Perfect
Alanoud Abumouti
Durrah Al Abduljabbar
Almozon Alamro
Layan Alkhalifah
Yara Alkhelaiwi
Raneem Almeshal
Nouf Almutairi
Raghad Alshammari
Fatimah Al Yousef
Salma Al Yousif

March 27, 2025

Contents

Chapter 1

Introduction

ReWork is a web-based application designed to support individuals in assessing their readiness to return to work after a period of mental health-related absence. The platform guides users through a reflective and supportive process that encourages emotional well-being and self-awareness during their journey back to employment. It primarily supports users through personalized programs and modules that contain helpful exercises, videos, and other resources to help them make meaningful progress. It also offers a safe space for users to reflect on their feelings, encouraging personal growth alongside practical progress.

The website is developed in collaboration with a client from the Institute of Psychiatry, Psychology, and Neuroscience. It serves as a valuable tool for applying and sharing academic expertise. By organizing content into structured programs and modules, the client is able to extend the reach of their work beyond traditional settings, while also refining it for use in their professional field. The system is built using the Django web framework and standard front-end technologies (HTML, CSS, JavaScript). It is accessible through modern desktop and mobile browsers and includes a content management interface for supplying content and organising it into structured pathways.

Chapter 2

Objectives and stakeholders

2.1 Project objectives

The system aims to serve two primary types of users: the **client** and the **end users**.

Client objectives

The client acts as the administrator of the platform, responsible for managing content and monitoring user engagement. The system is designed to deliver the following value to the client:

- A streamlined experience for managing digital learning content, including programs, modules, and resources.
- Tools to create, edit, and activate/deactivate **questionnaires** that assess users and guide them to personalized content.
- Full control over what content is published and how it is sequenced or categorized.
- The ability to view platform analytics, including enrollment trends and user progress.
- Access to **user-generated ratings and feedback**, allowing the client to make informed improvements to content and overall platform experience.

End user objectives

The end users are individuals who are aiming to re-enter the workforce and are seeking supportive resources. The platform helps them by:

- Providing a tailored onboarding experience through a dynamic questionnaire that identifies users' current situation and preferences.
- Ensuring users have easy access to the most relevant and supportive content that aligns with their individual needs and helps them progress confidently in their journey.
- Supporting self-assessment through interactive exercises, journaling features, and progress tracking tools.

- Creating a welcoming and intuitive interface that minimizes cognitive overload and encourages users to engage with the content at their own pace.

The overarching objective is to create a system that intelligently bridges the gap between the user's current situation and the resources that will most effectively help them move forward.

2.2 Stakeholder

2.2.1 Stakeholder analysis

The following stakeholder groups were identified:

- **Client (Administrator):** Manages all content and assessments on the platform. Their success depends on the usability of the backend tools and the quality of feedback they receive from the system.
- **End Users (Beneficiaries):** Use the platform to access helpful learning materials and track personal development. They rely on a responsive, non-overwhelming experience.
- **Developers and Designers:** Responsible for maintaining and improving the system. Their main interest is ensuring codebase sustainability and usability.
- **Organizational Partners or Funders:** May oversee the outcomes of the project or invest in it. Their interest lies in the platform's impact and adoption.

2.2.2 Key stakeholders

The two most critical stakeholders are:

- **The Client:** The project's success heavily relies on their ability to efficiently create and maintain relevant content. Their needs are central to the platform's backend design, especially around content flexibility and analytics.
- **The End Users:** The individuals navigating return-to-work pathways. Their satisfaction and outcomes directly determine the value of the project.

Chapter 3

Specifications

3.1 Functional specifications

This section outlines the core functionalities implemented across two Django apps: **users** and **client**. Each app is responsible for a distinct domain within the system and contributes to the overall goal of creating a personalised and trackable experience for users.

3.1.1 users App

The users app is responsible for user-related features such as account management, journaling, and personal motivation tools. These functionalities focus on supporting the user's personal development and engagement.

• User Registration and Authentication

Description: Allows users to register, log in, and manage account sessions securely.

Purpose: Enables user-specific experiences and data storage across sessions.

• User Profile Management

Description: Users have access to a profile page where they can view and update their personal information, including email and other editable details.

Purpose: Supports account customisation, helps keep user data accurate, and allows users to maintain control over their personal information.

• Post-Signup Questionnaire (Optional)

Description: After signing up, users are given the option to complete a questionnaire that assesses their readiness to return to work.

Purpose: This allows the system to provide personalised program recommendations based on the user's responses. However, completing the questionnaire is optional — users may instead choose to filter available programs and modules according to their preferences.

• Module and Program Enrollment

Description: Users can enroll in or unenroll from available modules/programs through the interface.

Purpose: Allows users to personalise their learning journey by selecting relevant modules/programs.

• **Module and Program Progress Tracking**

Description: Users can see how much of each module/program they have completed, with progress indicators.

Purpose: Encourages continued learning and provides insight into performance.

• **Interactive Module Content**

Each module provides the following user-accessible components:

1 - Exercises: Users can complete exercises to reinforce learning.

2- Videos: Educational videos are embedded for visual learning.

3- Resources: Supplementary materials such as PDFs, links, or guides are accessible.

4- Module Rating: Users can rate the module out of 5 starts and the average would appear on the module overview.

Purpose: These features together provide a holistic and engaging learning experience.

• **View Exercise Responses**

Description: Users' answers to module exercises are automatically saved. They can view their past responses at any time via the "Your Responses" section accessible from the navigation bar.

Purpose: Provides users with full control over their input, encourages reflection, and supports learning through revision and self-assessment.

• **Journal Entries**

Description: Users can write journal entries within a dedicated section of the app.

Purpose: Encourages reflection and supports personal growth by allowing users to freely express their thoughts.

• **Daily Affirmations**

Description: A daily affirmation is displayed in the user dashboard.

Purpose: Motivates users and enhances emotional engagement with the platform.

• **Dashboard Sticky Note**

Description: Users can write quick thoughts, daily tasks, or reminders on a sticky note that appears on their dashboard.

Purpose: Provides a lightweight and accessible way for users to stay organized and express day-to-day intentions, without needing to open the full journal. This promotes regular interaction with the platform and supports task management.

3.1.2 client App

The client app focuses on module management, progress tracking, and program interaction. It forms the learning and content delivery backbone of the system.

• User Management

Description: Clients can view a list of all registered users, see their personal details, and check which programs and modules they are enrolled in.

Purpose: Facilitates user support, oversight, and intervention where needed.

• Questionnaire Builder

Description: Clients can set up custom questionnaires by adding questions and grouping them into categories (e.g., confidence, readiness, skills).

Purpose: Supports gathering structured insights from users to personalize program recommendations and track change over time.

• Category Management for Assessments

Description: Client can create and edit categories that are used in questionnaires. These categories can also be associated with specific modules and programs.

Purpose: Categorisation supports a structured approach to user assessment and allows the system to better match users with programs and modules based on their responses in specific areas (e.g., confidence, skills, mindset). It also enhances reporting and content organisation.

• Program and Module Management

Description: Client can create, edit, and delete programs and modules. Each program can contain multiple modules, and each module can include exercises, videos, and additional resources.

Purpose: Enables the client to update the platform with relevant and tailored educational material.

• Exercises Management

Description: Client has access to a dedicated page where they can create and manage standalone exercises. Each exercise includes a question and can later be assigned to one or more modules as needed.

Purpose: Allows for reusability and consistency in assessment content across different modules. This also helps streamline the module creation process by enabling clients to pull from a pool of predefined exercises.

• Reporting and Analytics

Description: Client has access to a dashboard displaying key statistics in the form of interactive charts and graphs — covering module engagement, program participation, and user activity.

Purpose: Helps client monitor platform usage, evaluate effectiveness, and make data-informed decisions.

• **Data Export (CSV)**

Description: Client can export program/module/user statistics as downloadable CSV files.

Purpose: Enables offline analysis, external reporting, or integration with other systems.

3.2 Non-functional specifications

This section describes the non-functional requirements that define the quality attributes of the system. These requirements focus on how the platform operates rather than the specific functionalities it provides. They ensure the system is robust, secure, user-friendly, and capable of supporting both regular users and client effectively and efficiently over time.

• **Usability**

Description: The system is designed with a clean, intuitive user interface. Navigation is straightforward for both regular users (e.g., accessing modules, journals, and sticky notes) and the client (e.g., managing content and reports).

Purpose: To ensure users with various levels of digital literacy can interact with the platform confidently and without confusion.

• **Security**

Description: User data is protected through secure authentication (e.g., login, email verification), and access is role-based (users vs. client).

Purpose: To protect sensitive user information, including personal data, assessment responses, and progress records, from unauthorized access or tampering.

• **Reliability**

Description: The system maintains stability during regular usage and gracefully handles unexpected actions (e.g., invalid inputs or missing data).

Purpose: To ensure uninterrupted service and consistent behavior across different features and user interactions.

• **Performance**

Description: Pages load quickly, even when handling large datasets (e.g., when viewing user reports or filtering modules). Data exports (e.g., CSV reports) are processed efficiently.

Purpose: To maintain responsiveness and reduce waiting time, especially for client/admin users handling complex content.

• **Scalability**

Description: The architecture supports future growth, allowing more users, programs, modules, and data without major redesign.

Purpose: To accommodate long-term platform expansion, such as additional schools, institutions, or regional deployments.

- **Compatibility**

Description: The system works seamlessly with common tools used by clients, such as spreadsheet software (for CSV exports).

Purpose: To facilitate integration into existing workflows without requiring special tools or technical expertise.

- **Maintainability**

Description: The codebase is modular and well-documented, with a clear separation of concerns between apps (users and client).

Purpose: To allow for easy debugging, updates, and future development by different team members or contributors.

Chapter 4

Project Management

4.1 Introduction

This chapter reflects on the project management approaches used by the team during the development of the project. The team adopted an Agile methodology to manage the project, which allowed flexibility in responding to changing requirements and stakeholder feedback. This chapter evaluates the key decisions made, the challenges faced, and the lessons learned from the project management process.

4.2 Project Management Approaches Used

4.2.1 Initial Planning and Rationale

At the beginning of the project, the team decided to adopt an Agile approach due to its flexibility and ability to incorporate regular feedback. The initial project plan included:

- **Weekly meetings:** The team held one internal meeting every week to track progress and discuss challenges.
- **Client meetings:** We scheduled a separate weekly meeting with the client to gather feedback and clarify requirements.
- **Task tracking:** Tasks were managed using a Kanban board on Trello, which allowed the team to track progress and identify bottlenecks.

The rationale for choosing Agile was to maintain flexibility and adapt to evolving requirements as the project progressed. Since the project involved building a dynamic mental health intervention website, the iterative nature of Agile allowed the team to implement features incrementally and adjust based on client feedback.

4.2.2 Adjustments During the Project

Partway through the project, the team made several important adjustments:

- When the client went on a business trip, we increased the number of internal meetings to two per week to ensure better coordination within the team.

- We reduced the frequency of client meetings to once every two weeks during this period to accommodate the client's availability.
- After the client returned, we resumed weekly client meetings and continued to gather feedback regularly.

These adjustments helped the team maintain momentum despite the temporary unavailability of the client.

4.3 Major Decisions and Adjustments

4.3.1 Handling Ambiguous Requirements

One of the major challenges the team faced early on was the lack of clarity in the project requirements. Initially, the client seemed uncertain about the exact functionality and design of the website. Direct questioning did not resolve the issue, as the client was still forming a clear vision of the final product.

To address this, the team decided to:

- Create wireframes and mockups using **Figma**.
- Present the designs to the client to gather specific feedback and align expectations.
- Adjust the designs and features iteratively based on the input of the client.

This approach helped clarify the vision of the client and gave the team a concrete starting point for development.

4.3.2 Technical and Scheduling Challenges

The team also faced technical challenges during development:

- **Task Overload:** At one point, multiple high-priority tasks created a bottleneck.
- **Dependency Issues:** Some tasks were dependent on the completion of other components, which caused delays.

To resolve these issues, the team decided to:

- Reassign certain tasks to balance the workload more effectively.
- Introduce a clearer dependency tracking system on Trello to ensure that team members were aware of task dependencies.
- Prioritize critical tasks during stand-up meetings to prevent bottlenecks.

4.4 Challenges and Lessons Learned

4.4.1 Challenges Encountered

Apart from the initial ambiguity in requirements and technical issues, the team faced several other challenges:

- **Communication Gaps:** Initially, not all team members participated fully in meetings, leading to misunderstandings and inconsistent progress.
- **Client Availability:** The client's business trip created a temporary gap in direct feedback, which required the team to make independent decisions.
- **Scope Creep:** Midway through the project, the client requested additional features that were not part of the original scope.

4.4.2 How the Team Responded

The team addressed these issues by:

- Encourage active participation of all team members during meetings.
- Implementing a more structured feedback loop through Figma to align the team's work with the evolving requirements of the client.
- Managing scope creep by introducing a formal change request process, where new feature requests were evaluated against the project timeline and available resources.

4.4.3 Lessons Learned

The main lessons learned from the project were as follows:

- **Clearer Requirements Gathering:** Starting the project with more detailed user stories and requirements could have reduced early confusion.
- **Stronger Dependency Management:** Identifying and resolving task dependencies earlier would have improved efficiency.
- **Better Scope Management:** A formal process to manage new feature requests would have helped prevent scope creep.

4.5 Effectiveness of Agile Approach

Overall, the Agile approach was effective in providing flexibility and enabling the team to adapt to changing requirements. The iterative nature of Agile allowed the team to gather regular feedback and adjust the product accordingly. However, the lack of initial clarity in requirements and the dependency bottlenecks highlighted areas where more structure could have improved efficiency.

4.6 Conclusion

In summary, the team's use of an Agile approach allowed for flexibility and rapid adaptation to client feedback. Despite facing challenges with unclear requirements and task dependencies, the adjustments of the team — including more frequent meetings, clearer task tracking, and improved feedback loops — contributed to the successful completion of the project. In future projects, starting with more detailed requirements and implementing stronger dependency tracking would help improve overall efficiency.

Chapter 5

Design and Implementation

This chapter presents the architectural design and implementation decisions made during the development of our web-based educational platform. The system is built using Python and the Django web framework. It is divided into two primary Django apps: **users** and **client**. Each app is responsible for a distinct set of functionalities that align with the user and administrative workflows of the platform.

5.1 Technology Stack

The system was implemented using the following technologies:

- **Backend:** Python 3.13 with Django 5.1
- **Frontend:** HTML5, CSS3, JavaScript (with AJAX and jQuery in parts)
- **Database:** PostgreSQL (a powerful, production-ready relational database)
- **Version Control:** Git (GitHub)

The decision to use PostgreSQL was based on its robustness, scalability, and full feature set, making it suitable for production deployments and concurrent user access. It offers advanced data types, strong ACID compliance, and better performance over SQLite in multi-user environments.

The separation between frontend and backend logic was strictly enforced via Django's MVT (Model-View-Template) architecture, ensuring high maintainability and modularity.

5.2 Overall System Architecture

The platform adopts a modular and layered architecture, structured around the MVT (Model-View-Template) pattern. The architecture is divided into two main Django apps:

- **Users App:** Handles user authentication, registration, personal progress tracking, journals, sticky notes, questionnaires, and personalized module/program recommendations.

- **Client App:** Used by administrators to create and manage learning content (modules, programs, sections, videos, exercises, questionnaires), track user progress, and generate statistical reports.

This separation allows developers to focus on user-facing and admin-facing features independently, which enhances scalability and team collaboration.

5.3 Database Design

5.3.1 Core Models

The system's data model is comprehensive and normalized.

Note: PostgreSQL was used as the database engine for both development and production. This choice was driven by PostgreSQL's robust support for concurrent transactions, strong integrity constraints, and compatibility with Django's ORM. Its rich feature set allows for better long-term scalability, particularly as the platform grows in complexity and user base.

The core entities include:

- **User:** Extended from Django's AbstractUser model, with additional fields like gravatar, email verification, and profile linkage.
- **EndUser vs Admin:** Users are classified into `EndUser` (learners) and `Admin` (staff), stored separately to support RBAC (Role-Based Access Control).
- **Program & Module:** A program consists of ordered modules, implemented using a through-model (`ProgramModule`) to allow customizable sequencing.
- **Section & Exercise:** Reusable blocks of content, structured hierarchically for easy nesting and dynamic composition of learning material.

The models are highly relational and feature constraints like `unique_together` to ensure consistency in user progress tracking, questionnaire submissions, and ratings.

5.3.2 User Progress Models

User interactions with content are tracked in dedicated models:

- `UserModuleProgress`, `UserProgramProgress` track progress with percentage completion and status.
- `UserVideoProgress`, `UserExerciseProgress`, `UserResourceProgress` track granular progress inside modules.
- `ModuleRating` allows end users to rate each module once.

These models allow generating dynamic dashboards and analytics reports efficiently.

5.4 User Features (Users App)

5.4.1 Questionnaire-driven Recommendations

Upon registration, users complete a dynamic questionnaire to gather information about their background, preferences, and learning needs. Based on this, personalized program and module recommendations are generated using a simple matching algorithm on categories and sentiment-weighted answers.

Design Decision: We used categorical tagging on questions and matched user responses with module categories, prioritizing positive sentiment scores. This allowed lightweight recommendation logic without ML.

5.4.2 Sticky Notes and Journaling

Users can:

- Save personal sticky notes within the dashboard.
- Submit daily journal entries, tracking well-being factors like hydration, sleep, stress, and goal progress.

Non-functional requirement: Ensure that each user can only submit one journal per day, enforced via a `unique_together` constraint on the `JournalEntry` model.

5.4.3 User Dashboard

Users can track their:

- Module progress with completion bars.
- Recently accessed modules.
- Recommendations and daily quotes (using the `Quote` and `DailyQuote` models).

AJAX was used to power sticky note updates without page reloads.

5.5 Admin Features (Client App)

5.5.1 Module and Program Management

Admins can:

- Create/edit modules, sections, and exercises.
- Reuse sections/exercises across multiple modules.
- Reorder modules within a program via a drag-and-drop interface using JavaScript and AJAX.

Design Justification: Reusability was emphasized to minimize redundant content creation.

5.5.2 Questionnaire Builder

Admins can dynamically:

- Create questionnaires with rating/agreement scale questions.
- Tag questions with categories and sentiment.
- View and manage responder data with filters and pagination.

Alternative Considered: Hardcoded surveys vs dynamic builder. We chose dynamic generation to future-proof the system and allow research-driven question refinement.

5.5.3 Reports and Analytics

Admins can access:

- Visual analytics (Chart.js) on user activity, module ratings, and program enrollments.
- CSV exports of aggregated data for further analysis.

Non-functional emphasis: We optimized performance by using pre-aggregated queries, queryset caching, and pagination where needed.

5.6 Design Evolution

- Initially, we considered separating each model into a different Django app. We merged into ‘users‘ and ‘client‘ for clearer separation of roles.
- Module content editing was changed mid-development from forms to AJAX-powered inline updates, improving UX.
- Based on stakeholder feedback, journaling and video resources were added in later sprints.

Lesson Learned: We found that feedback loops and fast iteration significantly enhanced our product relevance and usability.

5.7 Security and Access Control

We used Django’s built-in decorators like `@login_required` and `@user_passes_test` to:

- Restrict access to admin dashboards.
- Ensure users only access their own data.
- Prevent duplicate submissions via model constraints.

5.8 Summary

The design and implementation decisions throughout the project were rooted in practical software engineering principles:

- Reusability and modularity in model relationships
- AJAX for seamless interactivity
- Enforced consistency via Django model constraints
- Extensibility to allow future feature growth (e.g., ML-based recommendations)

The final product reflects a clean, maintainable architecture that balances dynamic content delivery, real-time interactions, and data integrity.

Chapter 6

Testing

6.1 Approach and tools

To ensure that the software met its functional and non-functional requirements, the team used a combination of **automated testing**, **manual testing**, and **code quality tools**. Each approach served a distinct purpose in the overall testing strategy.

6.1.1 Automated Testing

Purpose: To verify that key features behave as expected and to ensure that code changes do not introduce regressions.

Tools Used:

Django's TestCase: Used for testing views and models, ensuring that database interactions and HTTP responses work as intended.

Coverage.py: To measure test coverage and identify untested paths.

Extent of Use: Most core logic and model behavior were covered with unit and integration tests. View functions that handle user requests (e.g., create$_{l}esson, assign_{t}utor)werealsotested.$

Evidence:

• Automated tests are located in the tests/ directories within each Django app.

• A coverage report (coverage$_{r}eport.html)isavailableintherootdirectory, showingthattheoveralltestcov$

6.1.2 Manual Testing

Purpose: To evaluate UI interactions and uncover usability issues or integration bugs not easily captured by automated tests.

Extent of Use: Relied on for testing all pages in login/signup, client dashboard, and users dashobard.

Evidence:

• Bug tracking and fixes can be found in git commits. For example: b3fec50, 72152f3, and 9b9707c.

6.2 Quality assurance processes

To ensure testing was thorough and consistent across the project, the team established a dedicated Quality Assurance (QA) sub-team, comprising half of the group members. This QA team was specifically responsible for maintaining test reliability, identifying test failures, and improving test coverage.

6.2.1 Dedicated QA Team

Purpose: The QA team was tasked with ensuring that all tests worked as intended and that the test suite meaningfully covered the system's functionality. Their work involved:

• Running all existing tests regularly to catch any regressions or failures.

• Investigating and fixing failing or broken tests. Monitoring code coverage across the project.

• Writing additional tests for files or modules with low coverage.

This approach allowed us to separate test maintenance and coverage tracking from feature development, ensuring dedicated focus on quality assurance without slowing down development progress.

Evidence:

• Git commit messages and tagged with qa: prefix (e.g., qa: fixed broken questionnaire test, qa: added stats tests to boost coverage) document improvements to test stability and coverage. See commits 9237f25 and 0cbe3e7.

• Code coverage tracking reports (generated via coverage.py) were shared and referenced to prioritize new test additions.

6.3 Evaluation of testing

6.3.1 Strengths

• High Code Coverage: Automated tests cover a significant portion of the codebase (99%), ensuring most logical paths are verified.

• Functional Testing Across Key Workflows: The team successfully tested complete user/-client journeys, such as enrolling in a module, creating a questionnaire, and verifying user emails. This ensured that key features worked together as intended in real-world use.

• Focused QA Effort: With a team assigned to testing, we ensured broken tests were fixed quickly and low-coverage areas received additional attention.

• Manual UI Testing: Manual testing helped identify edge-case bugs in forms, validations, and content rendering that automated tests may have missed.

6.3.2 Weaknesses

• Limited End-to-End Automation: Full user/client workflows were tested manually and with functional tests, but not automated using tools like Selenium or Cypress. These flows still require manual checking, which takes time and may lead to human error.

• Lack of Performance Testing: The system was not tested with many users or large amounts of data, so its behavior under heavy usage is unverified.

6.3.3 Suggestions for Future Improvement

• Automatic Test Runs: Use tools like GitHub Actions to run tests automatically whenever new code is added. This helps find problems early.

• Performance Testing: Incorporate basic load testing tools like Locust or JMeter to measure how the system handles large volumes of users and data.