VILNIUS UNIVERSITY

KAUNAS FACULTY

INSTITUTE OF SOCIAL SCIENCES AND APPLIED INFORMATICS

Study program Information Systems and Cyber Security

**Practice work 2**

**ULetter**

**Prepared by:**

Oleksandr Lanskiy

Vladyslav Nester

Arsenia Bohdanova

Yulia Lutsenko

Kseniia Klimenko

Kyrylo Kolesnichenko

**Checked by:**

dr. PhD Ilona Veitaitė

**Kaunas 2024**

Table of Contents

[1.INTRODUCTION 3](#_Toc185041782)

[2.RESEARCH 4](#_Toc185041783)

[2.1.Purpose 4](#_Toc185041784)

[2.2.Requirements 4](#_Toc185041785)

[2.3.Expectations 6](#_Toc185041786)

[3.PLANNING AND STRATEGY 7](#_Toc185041787)

[3.1.Team collaboration 7](#_Toc185041788)

[3.2.Technology stack 7](#_Toc185041789)

[3.3.System Diagrams 8](#_Toc185041790)

[4.DESIGNING AND CONTENT CREATION 11](#_Toc185041791)

[5.CODE AND DEVELOPMENT 12](#_Toc185041792)

[5.1.Backend Overview 12](#_Toc185041793)

[5.2.Frontend Overview 12](#_Toc185041794)

[5.3.Database Overview 13](#_Toc185041795)

[6.TESTING 15](#_Toc185041796)

[7.Notes 16](#_Toc185041797)

[7.1.How to Set Up the Project 16](#_Toc185041798)

[7.2.Application Development Lifecycle 17](#_Toc185041799)

[7.3.Website Requirements and Implementation 17](#_Toc185041800)

[Figure 1 Activity diagram 9](#_Toc185041614)

[Figure 2 Usecase diagram 9](#_Toc185041615)

[Figure 3 Sequence diagram 10](#_Toc185041616)

[Figure 4 Entity-Relationship Diagram 14](#_Toc185041617)

1. INTRODUCTION

This report presents the development of ULetter, an application created as part of the second practical assignment for the Web Development course. The project leverages the ChatGPT API to simplify the process of writing formal letters, aiming to assist users in creating meaningful and structured outputs with minimal effort. The report is structured into six main stages: research, planning and strategy, design and content creation, development, testing, and notes. Each stage documents the process, explains how the application was developed, and describes how the project works overall.

1. RESEARCH

In this part, we will provide more details about the web application, explaining what we decided during its development and the reasoning behind those choices. We will cover the purpose, requirements, and expectations for the application, laying the foundation for understanding its design and functionality.

* 1. Purpose

The purpose of the website is to provide users with a more comfortable and efficient way to write letters, simplifying the process and saving time. The application is designed to help users create formal and well-structured letters with minimal effort, aiming to serve as a practical and user-friendly tool. By streamlining the process, it ensures that users can focus on their message rather than struggling with formatting or tone.

The application is service-based, as it focuses on offering a functional service rather than simply presenting information or acting as a standalone product. This approach reflects its goal of providing consistent utility for letter-writing tasks. The primary target audience includes students, particularly those in their early years of university. This group often faces challenges with academic writing as they adapt to new expectations, such as crafting formal and professional letters. However, the application is not limited to students, as its versatile features make it suitable for a wide range of real-life communication needs, catering to anyone requiring professionally written letters across various contexts.

* 1. Requirements

We are expecting this application to meet significant demand, as it addresses a widespread issue: the challenges many individuals face in writing formal and structured letters. This is particularly relevant for students and others who require support in crafting professional communication. By providing a tool that simplifies this process and offers practical guidance, the application is likely to attract a wide range of users.

While the application is not strictly necessary for daily routines, its value lies in how much it simplifies users’ tasks. It enhances productivity, saves time, and improves efficiency, making it an important tool for those who frequently write letters. Users can benefit from a streamlined process that reduces the effort required to produce high-quality, formal letters.

The application is designed to meet key user needs, including reducing the burden of repetitive tasks, providing a cost-effective solution compared to other tools like ChatGPT, and ensuring broader accessibility. It targets users who are less familiar with advanced AI tools by offering a simplified and intuitive platform that caters to a wider audience.

Non-negotiable requirements for the application include ensuring reliability, delivering a user-friendly interface, and adhering to modern UI and UX design principles. The platform must be aesthetically pleasing, easy to navigate, and robust in its functionality. Avoiding bugs and providing a smooth, error-free experience are critical priorities to ensure user satisfaction and trust in the application.

Here are the **functional** and **non-functional requirements** for the project:

**Functional Requirements**:

1 The application must allow all users to create and edit letters.

2. It should enable users to save letter styles for future reference or reuse.

3. The system should support account creation and user authentication.

4. Users must specify the letter category and recipient to personalize the content.

5. The application should generate letters based on user input and preferences.

6. It must provide an option for users to edit their profile, including personal details and preferences.

7. The platform should allow users to review all previously saved styles for easy access.

8. Users should have access to advanced settings for more detailed letter specifications.

9. The system must maintain a comprehensive history of user requests and actions.

10. It should support downloading user profile information for personal records.

**Non-Functional Requirements**:

1. The application should have a responsive design, accessible on both desktop and mobile devices.

2. It must ensure fast response times, with API requests processed within a few seconds.

3. The platform should be reliable, with minimal downtime during operation.

4. It must adhere to modern UI/UX design principles for intuitive user interactions.

5. The system should provide secure data handling, including encryption for user information.

6. It must be scalable to handle a growing number of users and requests.

7. The application should implement CRUD operations, enabling create, read, update, and delete functionality.

8. It must be cost-efficient, keeping operational expenses reasonable.

9. It should provide comprehensive documentation for users and developers to ensure ease of understanding and maintenance.

10.The application must support regular updates and maintenance to adapt to changing user needs and technology trends.

* 1. Expectations

The expectations for this application are centered around providing an intuitive and user-friendly experience. The primary focus from a UI/UX perspective is visibility and simplicity, as the application is intended for frequent, possibly daily, use. It must be intuitively understandable and straightforward to navigate, ensuring users can effortlessly access its features and functionality without confusion.

The website’s primary function is to generate structured prompts and templates for the ChatGPT API. This simplifies the process of writing letters by offering pre-defined structures and categories that guide the user. Once the user provides input, the ChatGPT API generates the actual letter text based on the specified parameters, delivering a complete and well-crafted letter.

Users will expect features that align with the application’s purpose of simplifying the letter-writing process. These features may include saving and reusing styles, accessing predefined templates for different letter types, and leveraging advanced settings for more specific customization. Automation of repetitive tasks, such as formatting or content alignment, will further enhance the user experience by making the process quicker and more efficient.

In addition to the planned features, potential future enhancements could include incorporating tools for collaboration, enabling multiple users to work on the same letter draft, or adding integrations with external platforms for exporting letters directly to email or other applications. These additions would further broaden the application’s utility while staying aligned with its goal of simplifying and streamlining letter writing.

1. PLANNING AND STRATEGY

In the planning and strategy part, we will continue describing the project idea with a more focused approach, highlighting the key aspects such as the technology stack, content structure, and development strategy. This section will provide a clear outline of the project’s technical foundation, along with diagrams to simplify and enhance the understanding of the project concept and its planned implementation.

* 1. Team collaboration

The project is structured around a collaborative approach, with an emphasis on flexibility and shared effort, focusing solely on development and development-related tasks. Roles and responsibilities are limited to the technical aspects of the project, excluding considerations such as real marketing or distribution. While the project is formally divided into frontend and backend development, the practical implementation of tasks remains flexible, allowing team members to contribute based on their preferences and skills. This structure fosters a sense of ownership and adaptability within the team.

Quality assurance (QA) is also a key consideration, with certain members focusing on testing and ensuring the functionality and reliability of the application. However, the overarching project management philosophy avoids rigid role definitions, enabling dynamic participation across various aspects of the project. This strategy encourages collaboration and ensures that all team members have an opportunity to engage in areas they find most suitable.

To streamline workflow and task management, all project tasks have been organized and distributed using GitHub Projects. Each task is detailed and accessible to all team members, ensuring transparency and efficient collaboration. While we are not adhering to a specific software development methodology, our approach aligns with elements of the Big Bang model. This emphasizes rapid, iterative development without a heavy reliance on pre-defined structures, allowing for spontaneous contributions and adjustments as needed.

The report itself reflects this distributed approach, with some sections written collaboratively as a team while others are developed individually. This balance between collective and individual effort ensures a comprehensive understanding of the project, tailored to the team’s strengths and interests.

* 1. Technology stack

The technology stack for this project was chosen with a focus on balancing learning objectives and practical application. For the backend, we are using the Python Django framework, including Django REST framework. The choice of Django was straightforward, as Python offers versatility and usability in various domains beyond just web development, making it a more robust option compared to JavaScript backends. Given that this course emphasizes learning and preparing for future tasks, Python and Django were the most suitable choices for building the backend.

For the frontend, we opted for React, a JavaScript library widely regarded as one of the most popular and versatile tools for building user interfaces. This choice aligns with the course’s learning objectives, as React’s popularity and extensive ecosystem make it a valuable skill for future projects. Its component-based architecture also simplifies the development process, enabling better scalability and maintainability.

In terms of styling, we chose Tailwind CSS over traditional CSS. Initially, common CSS was considered, but we transitioned to Tailwind due to its simplicity and ease of use. Tailwind allows for rapid development and provides a more user-friendly experience when editing and managing styles. While it may not be as universally recognized as traditional CSS, in our experience, Tailwind significantly enhances productivity and makes writing code more intuitive.

For the database, we used SQLite, the default database provided by Django. Since the application is not planned for deployment or large-scale usage, SQLite was sufficient for our needs. Additionally, Django’s ORM (Object-Relational Mapping) facilitated seamless interaction with the database. If the need arises to scale the project for more advanced usage, Django’s backward compatibility makes it straightforward to transition to more robust database systems like PostgreSQL or MySQL. This flexibility ensures that the technology stack remains adaptable to future requirements.

* 1. System Diagrams

To provide a better understanding of the project’s structure and functionality, we created several diagrams. These include an Activity Diagram to showcase the workflow of the application, a Use Case Diagram highlighting the system’s main functionalities and user interactions, and a Sequence Diagram illustrating the flow of communication between components during specific processes. These diagrams collectively help to visualize the application’s design and operation more effectively.



Figure 1 Activity diagram

A diagram of a diagram

Description automatically generated

Figure 2 Use case diagram



Figure 3 Sequence diagram

1. DESIGNING AND CONTENT CREATION

The design process aimed to balance simplicity with functionality, ensuring that the application is both aesthetically pleasing and practical. By focusing on soft colors, we created a welcoming and professional environment that enhances readability and reduces visual strain. This choice reflects the application’s goal of simplifying letter writing and providing a distraction-free workspace for users.

Instead of relying on external tools like Figma, the design was directly integrated into the development workflow. This approach allowed for seamless adjustments between design and functionality, ensuring consistency and reducing potential miscommunication. All visual elements, such as buttons, menus, and typography, were designed to support smooth navigation and clear interaction.

Additionally, the design prioritizes accessibility and user experience. The layout is structured to guide users naturally through the application, presenting features and options in a logical order. Interactive components like menus and tabs were developed to be responsive and intuitive, ensuring a positive experience across different devices. Overall, the design phase emphasized creating a cohesive interface that aligns with the application’s purpose and the expectations of its users.

1. CODE AND DEVELOPMENT

In this part, we will describe the building process of the application, detailing how the designs and plans were transformed into a fully functional system. This includes the development of the frontend, backend, and database components, showcasing the technologies and methods used to bring the project to life.

* 1. Backend Overview

The backend development focused on creating a modular system using Django and Django REST framework. It consists of three main sub-applications: Users, Text Processing, and Style Personalization. The Users sub-application manages all user-related operations, such as retrieving, updating, and saving profile data. This part required more effort since the logic was implemented manually, using Django’s default features without relying on additional libraries.

The Style Personalization sub-application handles managing letter styles, allowing users to add, delete, and update them for customization. The Text Processing sub-application manages communication with the ChatGPT API, including applying filters and styles to user requests. Together, these sub-applications form the core functionality of the backend.

No external tools were used beyond the Django ecosystem, ensuring simplicity and making use of its built-in features like security and authentication. API testing was conducted using Postman to confirm that all endpoints worked as intended. Overall, the backend development prioritized creating a clean, scalable structure while ensuring reliability and adaptability for future improvements.

* 1. Frontend Overview

The frontend development began after the backend was completed, following a structured plan to ensure seamless integration between the two. Using React and Tailwind CSS, the frontend was designed to provide a responsive and user-friendly interface. The process involved translating the planned design into functional components, organized within the src directory for maintainability and scalability.

The frontend was developed with a clear focus on aligning it with the backend logic, ensuring smooth communication between the two. Once the initial frontend components were implemented, they were immediately integrated with the backend to test and refine the overall system. By sticking closely to the plan, the frontend effectively complements the backend, delivering a cohesive and intuitive experience for the users.

* 1. Database Overview

We were highly satisfied with the default database provided by Django, as it fully met our initial expectations. The pre-defined tables, such as the users table, were sufficient for our needs, and we did not encounter any reasons to extend or modify these tables. Additionally, we appreciated the automation provided by Django, which created necessary sub-tables automatically for certain functions, such as managing blacklisted tokens. This level of automation simplified the database management process significantly.

For custom implementations, we created additional tables to handle features like user history, styling preferences, and some basic user statistics. Throughout the process, we adhered to database normalization principles to ensure a clean and efficient structure. While most of the database’s functionalities were utilized, some tables, such as those for admin accounts, were left unused. This was primarily because the application does not require admin accounts, and we found local database management tools, such as DB Browser for SQLite, more convenient for our purposes.

Overall, the Django database proved to be a robust and adaptable solution for the project, enabling us to focus on functionality without excessive overhead. Below is the Entity-Relationship Diagram (ERD) for our database, which illustrates the relationships between the key entities used in the application.

A screenshot of a computer

Description automatically generated

Figure 4 Entity-Relationship Diagram

1. TESTING

We also paid significant attention to testing our project throughout the development process. The first stage of testing occurred during the development and implementation phases, where we focused on ensuring the functionality of all APIs and the application’s core logic. For backend testing during development, we relied on the debug mode provided by the Django server to track errors and verify the behavior of endpoints. Similarly, for frontend testing, we utilized browser developer tools to monitor responses and identify potential issues directly from the command line.

Additionally, Postman played a crucial role in validating the APIs. Its ease of use and extensive functionality made it an indispensable tool for testing and debugging the backend. With Postman, we systematically tested each endpoint to ensure proper communication between the server and client.

The second phase of testing involved writing formal test cases to confirm the application worked as intended. This included unit tests for backend features, such as user authentication, text processing, and style personalization, and API functionality tests. Specific tests validated the behavior of functions like user login, registration, password changes, and account deactivation, ensuring proper responses for both valid and invalid inputs.

Frontend tests focused on API integrations, ensuring the React components communicated effectively with the backend. For instance, authentication flows, user profile updates, and style management were all thoroughly tested. We also conducted tests to verify token handling, error responses, and other critical functionalities in the frontend.

1. Notes

In this part, we will highlight points that were either repeatedly referenced in previous sections or were not specifically covered elsewhere but are important to mention. This section includes additional details and insights that are considered valuable for understanding the overall project and its implementation.

* 1. How to Set Up the Project

To set up the project, ensure that Python and NPM are installed on your machine, as they are essential for running the backend and frontend components. Follow the steps below to get started:

1. Clone the Repository

• Clone the GitHub repository to your local machine.

2. Frontend Setup

• Navigate to the frontend directory in your terminal.

• Run the command npm install to install all required packages.

• Start the frontend server by running npm start.

• The server should automatically open a window in your default browser. If it doesn’t, manually enter the IP address of your local server into the browser to access it.

3. Backend Setup

• Navigate to the backend directory.

• Set up a virtual environment (venv) for Python and activate it.

• Run pip install -r requirements.txt to install all the backend dependencies.

• Set up the database by running the following commands in order:

1. python manage.py makemigrations

2. python manage.py migrate

• Start the backend server using the command python manage.py runserver.

4. ChatGPT Functionality Setup

• To enable the ChatGPT functionality, add a .env file in the project directory with your OpenAI API key in the format:

OPENAI\_API\_KEY=your\_key\_here

• (For testing purposes, my API key has been added in this version of the project so you can test the functionality directly. However, storing API keys in the project directory is not recommended and should be avoided for security reasons.)

By completing these steps, you will have both the backend and frontend servers running, and the project will be ready for use.

* 1. Application Development Lifecycle

Throughout the project, we did not follow a predefined or strict application development lifecycle methodology. Instead, we embraced a flexible approach, allowing team members to contribute in ways that best suited their individual strengths and interests. This created an environment that encouraged exploration and experimentation, fostering creativity and growth within the team.

If we were to align our process with a recognized methodology, it would closely resemble the Big Bang model. This approach reflects our focus on immediate development without extensive planning, allowing ideas and features to evolve naturally as the project progressed. While unconventional, this lifecycle approach supported our goals and aligned well with the team’s collaborative dynamics.

* 1. Website Requirements and Implementation

All requirements presented for the webpage were implemented, except for two that were deemed unsuitable for our specific application: the use of PHP sessions to process data on pages and the creation of at least one cookie. These requirements were initially included, but after evaluating the needs of our project and the technologies used, conscious decisions were made to approach these functionalities differently.

Since the project was built using Django instead of PHP, we initially considered implementing Django sessions as an alternative to PHP sessions. However, given the type of application and modern development standards, we decided to use JWT (JSON Web Token) authentication instead. JWT tokens provide a more modern, secure, and scalable method for managing user authentication and data exchange. Unlike traditional session-based approaches, JWT tokens offer stateless communication, aligning well with the goals and architecture of our application.

Regarding cookies, while they were considered for managing client-side state, we decided to implement local storage. Local storage was chosen as it allows for simpler implementation and avoids some of the security vulnerabilities commonly associated with cookies, such as cross-site scripting (XSS) attacks. This decision reflects current best practices for managing state in modern web applications and ensures a more secure and streamlined approach.