Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнтська частина)
Назва теми

КвРКІ 200115.20.01.14 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент IV курсу, група КІ2-20-1 _____ Б. О. Мазур
Підпис, Ініціали, прізвище

Керівник _____ В.М. Грига
Підпис, дата, Ініціали, прізвище

Нормоконтролер _____ І.О. Засорнова
Підпис, дата, Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис, Ініціали, прізвище

«14» червня 2024 р.

Хмельницький 2024

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра Комп'ютерної інженерії та інформаційних систем

Освітній рівень бакалавр

Галузь знань 12 Інформаційні технології

Спеціальність 123 Комп'ютерна інженерія

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

" 10 " 01 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Мазур Богдан Олегович

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнтська частина)

Керівник проекту (роботи) Грига В.М., д.т.н., проф.

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 15.02.2024 р. № 8

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2024 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Аналіз клієнтської частини існуючих програмно-технічних систем обробки, зберігання та передачі відеоконтенту

Архітектура системи та вибір програмних та апаратних засобів розробки клієнтської частини

Реалізація програмно-технічної системи

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Схема ієрархічної структури та взаємозв'язків компонентів клієнтської частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту

Схема взаємозв'язків та маршрутів між сторінками клієнтської частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту

Схема алгоритмічного забезпечення процесів аутентифікації, створення та відтворення відеоконтенту в рамках клієнтської частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Засорнова І.О., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання «_10_» 01 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2024	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2024	виконано
3	Робота над розділом 1 – аналіз клієнтської частини існуючих програмно-технічних систем обробки, зберігання та передачі відеоконтенту	01.03.2024	виконано
4	Робота над розділом 2 – архітектура системи та вибір програмних та апаратних засобів розробки клієнської частини	01.04.2024	виконано
5	Робота над розділом 3 – реалізація та оптимізація програмно-технічної системи	29.04.2024	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2024	виконано
7	Попередній захист ВКР	30.05.2024	виконано
8	Захист ВКР на засіданні ЕК	Червень 2024 року	

Студент  Б. О. Мазур.
Підпис Ініціали, прізвище

Керівник роботи  В. М. Грига.
Підпис Ініціали, прізвище

№ ряд ка	фор мат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 200115.20.01.14 ПЗ	Пояснювальна записка	66		
			Графічні матеріали			
2		КвРКІ 200115.20.01.14 Е8	Схема ієрархічної структури та взаємозв'язків компонентів клієнтської частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту	1		
3		КвРКІ 200115.20.01.14 Е8	Схема взаємозв'язків та маршрутів між сторінками клієнтської частини програмно- технічної системи обробки, зберігання та передачі відеоконтенту	1		
4		КвРКІ 200115.20.01.14 Е8	Схема алгоритмічного забезпечення процесів життєвого циклу користувача та відеоконтенту рамках в клієнтській частині програмно-технічної системи обробки, зберігання та передачі відеоконтенту	1		

Зм	Арк	№докум	Підпис	Дата	КвРКІ 200115.20.01.14 ВП		
Розробив		Мазур			Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Грига				У	1	1
Н. контр.		Засорінова				ХНУ, КІ2-20-1		
Затв.		Говорущенко		14.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнтська частина)».

Автор роботи: Мазур Богдан Олегович.

Керівник роботи: Грига Володимир Михайлович.

Пояснювальна записка: 66 с., 20 рис., 2 табл., 4 дод., 54 джерел.

Графічна частина: 3 креслення.

ПРОГРАМНО-ТЕХНІЧНА СИСТЕМА, ВІДЕОКОНТЕНТ, КЛІЄНТСЬКА ЧАСТИНА.

Метою дипломної роботи є розробка клієнтської частини програмно-технічної системи для обробки, зберігання та передачі відеоконтенту з клієнтської сторони. Оцінка механізмів обробки інформації у такій системі спрямована на забезпечення ефективного моніторингу об'єктів.

Об'єктом дослідження є функціонування клієнської частини програмно-технічної системи обробки відеоконтенту.

Предметом дослідження є оцінка різноманітних режимів застосування цієї системи для виявлення об'єктів у реальному часі.

Під час проведення даного дослідження був використаний метод систематичного огляду літератури для аналізу та вивчення предметної області програмно-технічної системи обробки, зберігання та передачі відеоконтенту.

Підпис студента

30.05.2024

Дата

ЗМІСТ

ВСТУП

В сучасному світі, що надзвичайно насичений технологіями, програмно-технічні системи відіграють ключову роль у різних сферах життя. Зокрема, важливою складовою цього технологічного прогресу є обробка, зберігання та передача відеоконтенту, що є важливим компонентом багатьох сучасних застосувань, починаючи від систем відеоспостереження і закінчуючи стрімінговими платформами для масового споживання контенту.

У даному дослідженні поставлено завдання вивчення та розробки програмно-технічної системи, спрямованої на оптимізацію обробки, зберігання та передачу відеоконтенту з клієнтської сторони. Ця система має на меті забезпечити ефективний моніторинг об'єктів у реальному часі, враховуючи сучасні вимоги до якості та швидкості обробки великих обсягів даних.

Зростання зацікавленості у використанні відеоконтенту у різних сферах діяльності вимагає розробки нових технологій, які забезпечували б не лише високу якість передачі та зберігання відео, але й ефективність обробки цих даних для подальшого аналізу та використання.

Метою даної дипломної роботи розробка клієнтської частини програмно-технічної системи для обробки, зберігання та передачі відеоконтенту з клієнтської сторони, а також оцінка різних режимів її застосування з метою ефективного моніторингу об'єктів. Об'єктом дослідження є функціонування цієї системи в реальних умовах, а предметом дослідження є оцінка її роботи у різних сценаріях використання.

Об’єктом дослідження є функціонування клієнської частини програмно-технічної системи обробки відеоконтенту.

Предметом дослідження є оцінка різноманітних режимів застосування цієї системи для виявлення об’єктів у реальному часі.

1 АНАЛІЗ КЛІЄНТСЬКОЇ ЧАСТИНИ ІСНУЮЧИХ ПРОГРАМНО-ТЕХНІЧНИХ СИСТЕМ ОБРОБКИ, ЗБЕРІГАННЯ ТА ПЕРЕДАЧІ ВІДЕОКОНТЕНТУ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

1.1.1 Опис предметної області

Відео контент і системи роботи з відео в сучасному світі є не лише розважальною платформою, але й потужним інструментом для комунікації, навчання та розвитку. Клієнтська частина цих систем відіграє ключову роль у забезпеченні зручності користувача та відтворенні відмінного користувацького досвіду.

У предметній області відео контенту клієнтська частина відображає інтерфейс, через який користувачі споживають відео. Це включає в себе елементи, такі як головна сторінка зі списком відео, сторінки з описом відео, коментарі, рекомендації та інше. Під час розробки аналогів YouTube, або аналогічної платформи, важливо враховувати потреби користувачів у зручності навігації, швидкості завантаження відео, а також у можливості взаємодії з контентом (лайки, коментарі, підписка на канал тощо).

Системи роботи з відео, такі як YouTube, Vimeo, або Twitch, володіють різноманітними функціональними можливостями. Наприклад, YouTube відомий своєю простотою використання та широким спектром функцій, включаючи можливість відтворення відео у високій якості, перегляд вмісту на різних пристроях та рекомендації відповідно до інтересів користувача. Vimeo, з іншого боку, часто використовується для публікації професійного відео контенту, тому має більше інструментів для контролю якості відео та взаємодії з аудиторією.

1.1.2 Виявлення актуальних проблем та завдань у цій сфері

Виявлення актуальних проблем та завдань у сфері клієнтського частини відео платформ визначається необхідністю вдосконалення користувацького досвіду та вирішення викликів, що виникають у процесі взаємодії з відео контентом. Нижче розглянуто поточні проблеми відео сервісів:

Повільне завантаження відео та інтерфейсу – завантаження великого обсягу відеоданих може призводити до тривалих затримок під час перегляду. Важливо вирішити проблеми оптимізації завантаження відео та інших компонентів фронтенду для поліпшення загального швидкодії та миттєвості реакції.

Не адаптованість інтерфейсу до різних пристроїв – з урахуванням різноманітності пристроїв, на яких може переглядатися відео, важливо забезпечити адаптивність інтерфейсу. Проблеми можуть виникнути при використанні мобільних пристроїв, планшетів або телевізорів, і вирішення цієї проблеми підвищить зручність користування [1].

Недостатня взаємодія з контентом – однією з ключових проблем може бути обмежена функціональність взаємодії з відео контентом. Розширення можливостей коментування, підписки та спільного перегляду може покращити залученість аудиторії та забезпечити позитивний взаємодійний ефект.

Проблеми з безпекою та конфіденційністю – в сучасному інтернет-середовищі, де дані користувачів мають велике значення, важливо забезпечити високий рівень безпеки та конфіденційності. Виявлення та вирішення потенційних загроз та проблем у цьому напрямку є ключовим завданням.

Низька доступність контенту для людей з обмеженими можливостями – для людей з різними обмеженнями (наприклад, відсутність зору або слуху) доступність відео контенту може бути обмеженою. Наприклад, відсутність адекватних альтернативних текстів для відео, субтитрів або аудіо дескрипцій. Розробка інтерфейсу, який дотримується стандартів доступності, є важливим завданням для забезпечення інклюзивного доступу до контенту.

Недостатня реактивність та відповідь інтерфейсу – швидка та реагуюча взаємодія з інтерфейсом є важливим аспектом зручного користування. Проблеми зі затримками у відгуку на дії користувача або несправності взаємодійних елементів можуть впливати на загальний враження від використання платформи.

1.1.3 Аналіз потреб користувачів та цільової аудиторії

Аналіз потреб користувачів та цільової аудиторії в контексті клієнтської частини відео платформ є ключовим етапом для створення успішного інтерфейсу, що задовольняє потреби широкого спектру користувачів.

Потреби щодо зручності навігації – користувачі очікують зручної та інтуїтивно зрозумілої системи навігації, яка дозволить їм легко знаходити відео, канали та інші функції платформи. Аналіз потреб у простому та ефективному пошуку, систематизації контенту та фільтрації результатів допоможе покращити навігаційний досвід [2].

Вимоги до адаптивності інтерфейсу – з огляду на різноманітність пристроїв, на яких може відбуватися перегляд відео, важливо врахувати потреби користувачів у адаптивному інтерфейсі. Це означає розробку інтерфейсу, який оптимально виглядає та працює на різних екранах та пристроях, включаючи комп'ютери, смартфони та планшети.

Потреби щодо спілкування та взаємодії – користувачі виявляють потребу у зручних засобах взаємодії з іншими користувачами та контентом. Це може включати в себе можливість коментувати відео, обмінюватися думками з іншими користувачами, а також спільний перегляд контенту. Аналіз цих потреб дозволить створити ефективні інструменти соціальної взаємодії.

Вимоги до доступності контенту – користувачі з різними фізичними та психічними обмеженнями мають потребу у доступному відео контенті. Це означає розробку інтерфейсу, який враховує стандарти доступності, такі як

підтримка субтитрів, аудіо дескрипцій та інших альтернативних методів сприйняття контенту.

Вимоги до персоналізації контенту – користувачі виявляють потребу у персоналізованому вмісті, який відповідає їхнім інтересам та вподобанням. Це може включати в себе рекомендації відео на основі переглянутих матеріалів, індивідуальні рекомендації каналів або підписок, а також персоналізовані рекомендації для покращення користувацького досвіду.

Потреби щодо безпеки та конфіденційності – користувачі мають вимоги щодо захисту своїх особистих даних та конфіденційності від третіх осіб. Це включає в себе захист від несанкціонованого доступу до облікових записів, захист від небажаних повідомлень та ефективні засоби контролю над власними даними.

Потреби у мобільності – з урахуванням зростаючої популярності мобільних пристроїв, користувачі мають вимоги до мобільності відео платформ. Це означає розробку мобільних додатків або адаптацію веб-інтерфейсу для зручного користування на смартфонах та планшетах.

Потреби у візуальній привабливості – користувачі очікують від відео платформи привабливого та сучасного дизайну, який стимулює їхній інтерес та створює позитивне враження. Це означає використання яскравих кольорів, привабливих графічних елементів та зручного розташування контенту на сторінці.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

1.2.1 Огляд існуючих систем роботи з відео

Розглянемо існуючі системи роботи з відео, з точки зору клієнтської частини, включає в себе вивчення їхнього інтерфейсу, функціональності та зручності користування для кінцевих користувачів. В нашому випадку ми розглянемо такі платформи, як: YouTube, Vimeo, Twitch, Dailymotion, Netflix, Amazon Prime Video, Apple TV+, Peacock, Crunchyroll, HBO Max, Disney+.

YouTube є однією з найбільш популярних платформ для спільного перегляду та завантаження відео. Його фронтенд вражає своєю простотою та інтуїтивно зрозумілим інтерфейсом. Головна сторінка пропонує персоналізований вміст, рекомендації та доступ до різноманітних функцій, таких як підписка на канали, коментування та взаємодія з іншими користувачами. YouTube також пропонує широкі можливості налаштування якості відео та роздільної здатності, щоб користувачі могли насолоджуватися контентом у відповідності з їхніми можливостями та умовами мережі .

Vimeo відомий своєю спрямованістю на професійний відеоконтент і має вишуканий фронтенд з акцентом на якість відео та креативний дизайн. Його інтерфейс дозволяє користувачам легко оглядати та відкривати відео, а також спілкуватися з авторами та іншими користувачами. Vimeo також відомий своєю підтримкою високих роздільних здатностей та інструментами для забезпечення якості відеоконтенту .

Twitch спеціалізується на стрімінгу відеоігор та живих трансляціях, і його фронтенд відрізняється від інших відеоплатформ. Інтерфейс Twitch пропонує широкі можливості для спілкування та взаємодії, таких як чати під відео, можливість підписки на стрімерів та участь у різноманітних інтерактивних елементах відтворення.

Dailymotion є ще однією популярною платформою для завантаження та перегляду відео. Його фронтенд пропонує зручний інтерфейс з можливістю швидко знаходити відео за різними категоріями, переглядати популярні відео та спілкуватися з іншими користувачами через коментарі та оцінки.

Netflix відомий своєю спрямованістю на стрімінг відео контенту, включаючи фільми, серіали та документальні фільми. Його фронтенд пропонує користувачам зручний інтерфейс з категоріями відео, рекомендаціями та можливістю створення персоналізованих списків перегляду. Користувачі також можуть оцінювати вміст, додавати коментарі та використовувати функцію автовідтворення для безперервного перегляду.

Amazon Prime Video є іншим популярним сервісом стрімінгового відеоконтенту, який пропонує широкий вибір фільмів, серіалів та оригінальних виробів. Його фронтенд включає зручний інтерфейс для пошуку та фільтрації контенту за різними категоріями та жанрами. Користувачі можуть створювати власні списки перегляду, додавати відгуки та використовувати функції автоматичного відтворення.

Apple TV+ є сервісом стрімінгового відеоконтенту, який пропонує оригінальні фільми, серіали та документальні фільми від компанії Apple. Його фронтенд вражає своєю простотою та елегантністю, забезпечуючи зручний інтерфейс для пошуку та відтворення контенту. Користувачі можуть насолоджуватися широким вибором жанрів та категорій, а також скористатися різноманітними функціями, такими як автоматичне відтворення наступного епізоду, завантаження вмісту для перегляду офлайн та спільний перегляд з друзями через AirPlay .

Peacock є сервісом стрімінгового відеоконтенту від NBCUniversal, який пропонує широкий вибір фільмів, серіалів, спортивних подій та оригінальних програм. Фронтенд Peacock надає користувачам зручний інтерфейс для перегляду контенту, де вони можуть знаходити вміст за різними категоріями та жанрами. Крім того, користувачі можуть скористатися різноманітними функціями, такими як прискорений перегляд, автоматичне відтворення наступного епізоду та перегляд живих телевізійних каналів.

Crunchyroll є популярним сервісом стрімінгового відеоконтенту для аніме та манґи. Його фронтенд пропонує широкий вибір аніме-серіалів та фільмів, які доступні для перегляду у високій якості. Користувачі можуть швидко знаходити вміст за різними жанрами та категоріями, додавати аніме до черги перегляду та користуватися різними функціями спільного перегляду та обговорення вмісту з іншими користувачами.

HBO Max є сервісом стрімінгового відеоконтенту від HBO, який пропонує широкий вибір фільмів, серіалів та оригінальних програм. Фронтенд HBO Max

включає зручний інтерфейс для пошуку та перегляду контенту, а також можливість створення власних списків перегляду та персоналізованих рекомендацій. Користувачі можуть також користуватися різними функціями, такими як прискорений перегляд, відтворення вмісту у високій якості та спільний перегляд з друзями.

Disney+ є популярним сервісом стрімінгового відеоконтенту, який спеціалізується на вмісті від компанії Disney, включаючи фільми, серіали, анімаційні фільми та документальні програми. Його фронтенд вражає простотою та зручністю використання, пропонуючи користувачам широкий вибір категорій та рекомендацій на основі перегляду. Користувачі можуть також створювати власні профілі, додавати контент до списку відтворення та користуватися різними функціями, такими як автоматичне відтворення та скачування вмісту для перегляду офлайн.

1.2.2 Порівняння їх функціональних можливостей, дизайну

Проведемо порівняльний аналіз функціональних можливостей та дизайну різних відео платформ з точки зору клієнтської частини, що дасть можливість краще зрозуміти їхні переваги та недоліки, а також визначити кращі практики для розробки нової платформи.

YouTube відзначається широким вибором функціональності, включаючи можливість завантажувати, коментувати та вподобати відео, створювати плейлисти та підписуватися на канали. Також він має простий та чистий дизайн з легким доступом до основних функцій, таких як пошук відео, підписка на канали та налаштування відтворення.

Vimeo славиться своєю якістю відео та функціональністю для творців вмісту, такою як можливість встановлення пароля для відео, збір зборів та виведення вмісту на преміумних плейерах. Дизайн відрізняється елегантним та

мінімалістичним дизайном, який підкреслює якість відеоконтенту та зручність навігації

Twitch відомий своєю спеціалізацією на стрімінгу відеоігор та включає в себе унікальні функції спільного перегляду та чату для глядачів. У дизайні використано яскраві кольори та елементи геймінгу у своєму дизайні, що відображає специфіку платформи для стрімінгу відеоігор

Dailymotion пропонує широкий вибір відеоконтенту та функціональність, яка подібна до YouTube, з підтримкою плейлистів, коментарів та підписки на канали. Дизайн має сучасний та привабливий дизайн, який робить навігацію по сайту легкою та приємною для користувачів

Також додатково можна переглянути порівняльну таблицю 1.1, де наведено узагальнену оцінку функціональних можливостей та дизайну кожного сервісу з точки зору користувача, що допомагає порівняти їх.

Таблиця 1.1 – Порівняння характеристик відомих відео сервісів

Характеристика	Інтерфейс	Пошук та навігація	Рекомендації	Соціальна взаємодія
1	2	3	4	5
YouTube	Простий, інтуїтивний	Ефективний	Так	Так
Vimeo	Стильний, елегантний	Зручний	Так	Так
Twitch	Ігровий, динамічний	Персоналізований	Ні	Так
Dailymotion	Простий, зручний	Простий, ефективний	Так	Так
Netflix	Стильний, сучасний	Ефективний	Так	Ні

Кінець таблиці 1.1 – Порівняння характеристик відомих відео сервісів.

1	2	3	4	5
Hulu	Зручний, простий	Зручний	Так	Так
HBO Max	Простий, інтуїтивний	Ефективний	Так	Так
Disney+	Простий, елегантний	Ефективний	Так	Ні
Crunchyroll	Простий, зручний	Зручний	Так	Так
Apple TV+	Простий, елегантний	Зручний	Так	Ні
Peacock	Зручний, простий	Зручний	Так	Так
YouTube TV	Зручний, простий	Ефективний	Так	Ні
Amazon Prime Video	Зручний, простий	Зручний	Так	Ні

1.3. Методологічні підходи до вирішення задачі за темою дослідження

1.3.1 Огляд наукових методів та підходів до розробки систем роботи з відео

Розглянемо основні наукові методи та підходи до розробки систем роботи з відео з точки зору клієнтської частини, давайте розпочнемо з дослідження користувацьких потреб:

1. Методи емпатії, такі як спостереження за користувачами, інтерв'ю та анкетування, допомагають зрозуміти потреби та вимоги користувачів від фронтенду відео платформи [3].

2. Аналіз використання, тестування користувачів та збір зворотного зв'язку дозволяють виявити сильні та слабкі сторони існуючого фронтенду та визначити області для подальшого вдосконалення.

Принципи дизайну користувацького інтерфейсу (UI) та дослідження користувацького досвіду (UX):

1. Застосування принципів UI/UX дизайну, таких як простота, зручність, консистентність та доступність, сприяє створенню інтуїтивно зрозумілого та привабливого фронтенду.

2. Аналіз потреб та поведінки користувачів у процесі взаємодії з фронтендом дозволяє ідентифікувати ключові аспекти, які впливають на їхній досвід використання.

Технології розробки клієнтської частини:

1. Використання сучасних технологій, таких як HTML, CSS, JavaScript та їхні фреймворки (наприклад, React, Angular, Vue.js), дозволяє створювати динамічні та інтерактивні користувацькі інтерфейси.

2. Врахування вимог до швидкості завантаження та реагування клієнтської частини допомагає забезпечити плавну та ефективну роботу платформи для користувачів.

Тестування та валідація:

1. Використання методів тестування, таких як A/B тестування, тестування з користувачами та автоматизоване тестування, допомагає перевірити функціональність та ефективність фронтенду перед випуском його в продакшн .

2. Здійснення регулярних оновлень та вдосконалень на основі зібраного зворотного зв'язку допомагає підтримувати високу якість користувацького досвіду та відповідати змінюваним потребам користувачів.

1.3.2 Вибір та обґрунтування методології дослідження

При виборі методології дослідження для розробки клієнстької частини відео платформи важливо врахувати специфіку проекту та потреби користувачів.

Існують такі методології та їх обґрунтування з точки зору клієнтської частини або фронтенду:

1. Design Thinking - це методологія, спрямована на розв'язання складних проблем з орієнтацією на потреби користувачів. Вона дозволяє розглянути розробку клієнської частини з точки зору користувача, акцентуючи увагу на їхніх потребах та очікуваннях [4].

2. Lean UX - це методологія, яка покликана зменшити час розробки та випуску продукту за рахунок швидкої ітерації та залучення користувачів у процес розробки [5].

3. Agile - це ітеративний підхід до розробки програмного забезпечення, який спрямований на гнучкість, швидкість та взаємодію учасників команди.

4. User-Centered Design (UCD) - це підхід до розробки, який зосереджений на користувачах та їхніх потребах на кожному етапі процесу [6].

Обираючи методологію дослідження, слід враховувати велику кількість потреб користувачів; в свою чергу, використовуватиметься комбінація всіх вище перечислених методологій. Комбінація кількох методологій або їх адаптація до конкретних умов може бути найбільш ефективним підходом до розробки клієнтської частини відео платформи.

1.3.3 Опис методів збору та аналізу даних

Збір та аналіз даних - це потужний інструмент, який дає нам можливість ґрунтувати свої рішення на фактах, а не на інтуїції, економити час та ресурси, краще розуміти своїх клієнтів, розробляти нові продукти та послуги, а також покращувати життя людей.

Дані допомагають нам виявляти закономірності, передбачати майбутні події, оптимізувати роботу, отримувати нові знання, збільшувати прибутки та краще розуміти світ навколо нас. Аналіз даних використовується в багатьох сферах, таких як бізнес, охорона здоров'я, уряд та наука.

Збір та аналіз даних - це постійно зростаюча сфера, яка має величезний потенціал для покращення нашого життя.

Основні методи збору та аналізу даних з точки зору клієнтської частини, які допомагають зрозуміти, як користувачі взаємодіють із системою та як можна покращити їхній користувацький досвід:

1. Аналіз використання (User Analytics) – використання інструментів аналізу використання, таких як Google Analytics або Hotjar, дозволяє збирати дані щодо того, як користувачі взаємодіють з різними елементами фронтенду. Важливі метрики включають кількість відвідувань, час, проведений на сторінці, шляхи навігації та взаємодії з різними елементами інтерфейсу [7].

2. Збір зворотного зв'язку від користувачів – включення механізмів збору зворотного зв'язку, таких як опитування, форми зворотного зв'язку або кнопки "Повідомити про проблему", дозволяє користувачам висловлювати свої враження та зауваження стосовно фронтенду. Аналіз такого зворотного зв'язку може допомогти ідентифікувати проблеми та пропозиції щодо покращень.

3. Тестування користувацького досвіду (User Experience Testing) – проведення тестувань користувацького досвіду, таких як A/B тестування або тестування з учасниками, дозволяє визначити, як різні варіанти фронтенду впливають на сприйняття та взаємодію користувачів. Аналіз результатів таких тестів може вказати на оптимальні шляхи оптимізації фронтенду для досягнення кращого користувацького досвіду [8].

4. Моніторинг помилок та відгуків – використання інструментів моніторингу помилок, таких як Sentry [9] або Rollbar [10], дозволяє виявляти та вирішувати проблеми, які можуть виникнути при використанні клієнської частини користувачами. Аналіз відгуків та оглядів на платформах, таких як App

Store або Google Play, може також вказати на конкретні аспекти фронтенду, що потребують уваги.

Комбінація цих методів дозволяє отримати повний обсяг даних щодо використання клієнтської частини користувачами, їхніх проблем та потреб. В свою чергу я буду використовувати комплексний підхід для максимально точного аналізу та подальшого вдосконалення.

1.4 Постановка задачі

1.4.1 Формулювання мети та завдань

Основна мета даного дослідження полягає в тому, щоб ретельно проаналізувати користувацький досвід та вимоги до клієнтської частини відео платформи з метою покращення її функціональності, ефективності та зручності використання, у власній платформі. Це дослідження спрямоване на забезпечення максимально задоволення потреб користувачів та підвищення конкурентоспроможності платформи на ринку.

Для досягнення цієї мети були визначені наступні завдання дослідження:

1. Аналіз потреб користувачів – дослідити та зрозуміти потреби та очікування користувачів відеоплатформи, включаючи їхні вподобання щодо функціональності, дизайну та взаємодії з інтерфейсом.

2. Оцінка користувацького досвіду – проаналізувати існуючий користувацький досвід використання фронтенду відео платформи з метою виявлення сильних та слабких сторін, а також можливостей для покращення.

3. Визначення ключових аспектів функціональності – встановити основні функціональні вимоги до клієнтської частини відео платформи, які забезпечать зручний та ефективний користувацький досвід.

4. Розробка оптимального інтерфейсу – розробити імплементацію фронтенду, яка відповідає виявленим потребам та очікуванням користувачів, з урахуванням сучасних тенденцій у дизайні та функціональності.

5. Тестування та валідація нового інтерфейсу – провести тестування нового інтерфейсу з учасниками, щоб перевірити його ефективність, зручність та відповідність потребам користувачів.

6. Аналіз результатів та вдосконалення – оцінити результати тестування та отриманий зворотний зв'язок від користувачів для ідентифікації можливостей для подальшого вдосконалення фронтенду відео даної платформи.

1.4.2 Визначення чітких та вимірних критеріїв оцінки результатів

Для ефективної оцінки результатів дослідження та розробки клієнтської частини відео платформи важливо визначити чіткі та вимірювані критерії успішності. Ці критерії повинні відображати досягнення поставленої мети та завдань дослідження і дозволити об'єктивно оцінити результати.

Перелік основних критерії оцінки:

1. Покращення користувацького досвіду – метрики змін у показниках задоволеності користувачів, вимірювані через анкети, опитування та інші методи збору зворотного зв'язку та вимір збільшення кількості задоволених користувачів, зменшення кількості скарг або запитів на підтримку.

2. Покращення ефективності взаємодії – час, витрачений користувачами на досягнення своїх цілей, такий як пошук і перегляд відео та вимір зменшення середнього часу на виконання основних завдань користувачів, наприклад, знаходження відео та його перегляд.

3. Підвищення залученості користувачів – метрики збільшення кількості активних користувачів, зміна в рівні взаємодії зі сторінками та функціоналом платформи та вимір зростання кількості відвідувань, кількість сеансів користувачів та кількість взаємодій на сторінках.

4. Покращення конверсії – відсоток користувачів, які виконують цільові дії, такі як реєстрація або підписка на вміст та вимір зростання конверсійної

швидкості, зменшення відпадання користувачів на критичних етапах, підвищення кількості цільових дій.

5. Покращення стабільності та продуктивності – час завантаження сторінок, кількість помилок та збоїв, продуктивність роботи фронтенду під навантаженням.

Зазначені критерії дозволять виявити досягнення у вдосконаленні клієнтської частини відео платформи. Вони є вимірними та об'єктивними, що дозволяє зробити оцінку результатів дослідження та розробки максимально точною і ефективною.

1.4.3 Обґрунтування актуальності та практичної значущості дослідження

Основним питанням даного дослідження є розробка та оптимізація клієнтської частини відеоплатформи з метою покращення користувацького досвіду, це є ключовим чинником успіху будь-якої онлайн-системи. У відповідний спосіб побудований фронтенд може значно вплинути на сприйняття платформи користувачами, їхню активність та лояльність.

Дослідження актуальне у зв'язку з постійним розвитком інтернет-технологій та зростанням конкуренції на ринку відео платформ. Користувачі стають все вимогливішими щодо зручності використання, якості контенту та можливостей платформи. Перевага над конкурентами може бути досягнута саме за рахунок вдосконалення клієнтської частини, що робить це дослідження важливим для подальшого успіху.

Практична значущість дослідження проявляється у здатності вдосконалити користувацький досвід та покращити функціональність веб-додатку, що безпосередньо вплине на залученість користувачів та їхню лояльність. Результати дослідження допоможуть побудувати більш ефективний фронтенд, який буде відповідати потребам та очікуванням користувачів, забезпечуючи зручний та приємний користувацький досвід. Лише завдяки ретельному аналізу та

дослідженню можна забезпечити відповідність платформи сучасним стандартам та очікуванням користувачів.

1.5 Висновки

У межах розділу 1 було проведено теоретичний аналіз, вивчено недоліки існуючих рішень та сформульовано завдання на дослідження шляхів їх вирішення. Перший етап роботи передбачав ознайомлення з предметною областю, включаючи структуру та базову модель організації. На основі проведених досліджень було визначено основні функції, які має виконувати клієнтська частина відео платформи.

Мета дослідження, яка полягала в розробці та оптимізації клієнтської частини веб-платформи з метою покращення користувацького досвіду, була досягнута. На основі аналізу та експериментів було розроблено ряд рекомендацій щодо вдосконалення інтерфейсу та функціональності платформи.

2 АРХІТЕКТУРА СИСТЕМИ ТА ВИБІР ПРОГРАМНИХ ТА АПАРАТНИХ ЗАСОБІВ РОЗРОБКИ КЛІЄНТСЬКОЇ ЧАСТИНИ

2.1 Вибір програмних засобів

Вибір операційної системи відіграє критичну роль у процесі розробки та подальшої експлуатації програмно-технічної системи для обробки, зберігання та передачі відеоконтенту. Для розробки цього проєкту використовується операційна система Windows, яка є популярним вибором серед розробників завдяки своєму зручному інтерфейсу, широкому набору доступних інструментів та програм для розробки, а також великій підтримці з боку спільноти. Windows надає розробникам зручні умови для ефективної роботи з кодом, тестування та дебагінгу програм, що є ключовими етапами розробки будь-якого проекту [11].

Для самої системи обрано операційну систему Linux, яка є відкритою, надійною та ефективною платформою для розгортання серверних застосунків. Linux вирізняється високим рівнем безпеки, стабільністю та гнучкістю у налаштуванні, що робить її ідеальним вибором для систем, що вимагають безперервної роботи та обробки великих обсягів даних, як це і є у випадку з системами обробки відеоконтенту. Використання Linux дозволить оптимізувати роботу системи, забезпечити її стабільне та ефективне функціонування на серверах, а також спростити процес управління ресурсами і розгортання додатків [12].

Такий вибір операційних систем забезпечує баланс між зручністю та ефективністю розробки в середовищі Windows та надійністю, безпекою та високою продуктивністю запуску та експлуатації системи на Linux. Це дає змогу використовувати переваги обох операційних систем для досягнення найкращих результатів у проекті.

В свою чергу вибір мов програмування є ключовим аспектом при розробці програмно-технічної системи, особливо коли йдеться про клієнтську частину, яка вимагає високої продуктивності, швидкості реакції інтерфейсу та зручності

взаємодії для користувача. Для цього проекту було обрано поєднання TypeScript і JavaScript, які разом формують потужний інструментарій для розробки клієнтської частини.

JavaScript є універсальною мовою програмування, яка домінує у сфері веб-розробки вже багато років. Вона дозволяє створювати динамічні веб-сторінки, що можуть взаємодіяти з користувачем без необхідності перезавантаження сторінки. Ця мова є невід'ємною частиною клієнтської розробки та підтримується усіма сучасними веб-браузерами, що робить її ідеальним вибором для створення кросплатформних додатків [13].

TypeScript, у свою чергу, є надмножиною JavaScript, що додає строгу типізацію та об'єктно-орієнтовані можливості, які допомагають у великих та складних проектах. Використання TypeScript сприяє підвищенню продуктивності розробника за рахунок кращої підтримки в редакторах коду, автоматичного виявлення помилок на етапі компіляції та спрощення рефакторингу. Таким чином, TypeScript дозволяє писати більш надійний код, зменшує ймовірність виникнення помилок у рантаймі та спрощує роботу в команді завдяки чітким контрактам між компонентами системи [14].

Комбінація JavaScript і TypeScript у проекті надає змогу поєднати гнучкість та широкі можливості JavaScript з перевагами строгої типізації та об'єктно-орієнтованого програмування, які надає TypeScript. Цей підхід забезпечує високу ефективність розробки, легкість підтримки коду та його масштабування, що є важливим для створення сучасних веб-додатків та інтерактивних інтерфейсів.

Не менш важливим є вибір бібліотек та фреймворків є критичним рішенням у процесі розробки програмного забезпечення, особливо коли йдеться про клієнтську частину, де вимоги до швидкості, адаптивності та користувацького досвіду є вищими, ніж коли-небудь. У цьому проекті для розробки веб-додатків додатків було обрано фреймворки Next.js [15] відповідно, який базується на React.js [16] – сучасній бібліотеці для створення користувацьких інтерфейсів від Facebook.

Next.js є популярним фреймворком для розробки веб-додатків на React, який надає можливості серверного рендерингу, що значно покращує час завантаження сторінок та оптимізує їх для пошукових систем. Це особливо важливо для комерційних веб-сайтів та платформ, де висока видимість у пошукових системах та швидке завантаження є критичними для залучення та утримання користувачів. Крім того, Next.js спрощує процес розробки завдяки вбудованим функціям маршрутизації, оптимізації зображень та генерації статичних веб-сторінок, що дозволяє розробникам зосередитися на бізнес-логіці, не турбуючись про базову інфраструктуру.

Обрання Next.js для клієнтської частини проекту забезпечує потужний інструмент для розробки високоякісних веб-додатків. Це дозволяє спростити підтримку та оновлення продуктів, а також сприяє швидкій ітерації продукту та його адаптації до змінюваних вимог ринку та користувачів.

2.2 Вибір апаратних засобів

Вибір процесора для системи, особливо коли йдеться про серверне обладнання для хостингу додатку, є фундаментальним рішенням, що впливає на продуктивність, надійність та масштабованість проекту. В ідеалі, серверний процесор повинен забезпечувати високу обробну потужність для обслуговування великої кількості одночасних запитів без втрати продуктивності, а також мати високу енергоефективність для зниження витрат на електроенергію та охолодження в дата-центрі.

У цьому контексті, серії процесорів, такі як AMD EPYC [17] та Intel Xeon [18], є відмінними кандидатами для серверів, що вимагають високої продуктивності. AMD EPYC вирізняється завдяки своїм мультиядерним конфігураціям, що забезпечують значну обчислювальну потужність, а також підтримці великої кількості одночасних потоків, що ідеально підходить для вимог сучасних веб-додатків та баз даних. З іншого боку, Intel Xeon продовжує бути

лідером у галузі завдяки своїй надійності, підтримці розширеного набору інструкцій та високій продуктивності у різноманітних обчислювальних навантаженнях.

Для проєкта, який потребує серверного хостингу клієнтської частини, важливо також враховувати фактори, такі як підтримка віртуалізації, можливість швидкої масштабованості та легкість інтеграції з існуючою інфраструктурою дата-центру. Обидва варіанти, AMD EPYC і Intel Xeon, пропонують рішення, які можуть бути оптимізовані під конкретні потреби проекту, включаючи специфікації щодо кількості ядер, частоти, пам'яті та кешу, що дозволяє досягти оптимального балансу між ціною та продуктивністю.

У даному випадку, вибір впав на користь процесора Intel XEON 4 Core E3-1225 V5 3.30GHz (SR2LJ) [19], що є цілком виправданим, враховуючи специфічні потреби сервера, задіяного в хостингу веб-додатків. Серія Intel XEON відома своєю високою продуктивністю, надійністю та енергоефективністю, що робить її ідеальним вибором для серверних рішень.

Процесор E3-1225 V5, зображено на рисунку 2.1, він пропонує 4 ядра з базовою частотою 3.30GHz, що забезпечує високий рівень обчислювальної потужності, необхідної для обробки запитів до веб-додатку та виконання фонових задач, таких як обробка даних, забезпечення безпеки та інші сервісні процеси. Також важливою перевагою є підтримка технологій віртуалізації, які дозволяють оптимізувати використання ресурсів сервера шляхом розділення фізичного сервера на кілька віртуальних машин. Це дозволяє ефективно масштабувати додатки, оптимізуючи використання серверних ресурсів під зростаючі або змінні навантаження.

Крім того, Intel XEON E3-1225 V5 підтримує розширені технології безпеки та керування, що є критично важливим для забезпечення надійності та доступності хостингового середовища. Такі функції, як Intel vPro, дозволяють дистанційно керувати сервером, здійснювати моніторинг стану апаратного

забезпечення та вживати заходів безпеки без необхідності фізичного доступу до обладнання.

Рисунок 2.1 – Зовнішній вигляд процесора Intel XEON E3-1225 V5

Враховуючи ці аспекти, Intel XEON E3-1225 V5 є оптимальним вибором для сервера, що забезпечуватиме хостинг веб-додатку. Його обчислювальна потужність, надійність та підтримка сучасних технологій забезпечать стабільну та ефективну роботу додатка, гарантуючи високу доступність та безперебійність обслуговування користувачів.

Наступним етапом є вибір графічного процесора (GPU) [20] для сервера, що використовуватиметься для хостингу додатку, зазвичай зумовлюється специфічними потребами обробки даних, такими як обчислення великої кількості даних або задачі машинного навчання. Проте, в контексті задач, де GPU потрібен лише для підключення та налаштування серверу, вимоги до графічного процесора значно нижчі. В такому випадку, важливими факторами вибору стають енергоефективність, надійність та сумісність із серверною платформою, а не

висока обчислювальна потужність, яка необхідна для інтенсивних графічних або обчислювальних задач.

Для базових потреб підключення та налаштування сервера оптимальним вибором може бути інтегрований графічний процесор або базовий дискретний GPU, який забезпечує достатню продуктивність для запуску операційної системи, виконання стандартних серверних налаштувань та обслуговування дистанційного доступу без необхідності вкладень у високопродуктивні графічні процесори. В цьому контексті ідеальним варіантом є використання відеокарти NVIDIA QUADRO K420/K600 1GB High Profile [21], зображено на рисунку . Вибір скромнішого GPU дозволяє знизити загальні витрати на серверну інфраструктуру, одночасно забезпечуючи необхідну функціональність для адміністрування та підтримки сервера.

Рисунок 2.2 – Зовнішній вигляд дискретної відеокарти nVidia Quadro K420, 2 GB GDDR3, 128-bit / DVI, DisplayPort

Також, враховуючи, що GPU не буде використовуватися для вимогливих задач, таких як візуалізація даних, ігри або обробка відео в реальному часі, можна відмовитися від дорогих рішень на користь більш економічних варіантів, що спрощує процес вибору та інтеграції GPU в серверне обладнання. Вибір такого

рішення дозволяє зосередитися на ключових аспектах серверної продуктивності, таких як обробка запитів, управління даними та забезпечення високої доступності хостингових послуг, без зайвих витрат на невикористовувані графічні ресурси.

Далі потрібно вибрати обєм оперативної пам'яті для сервера, який буде використовуватися для хостингу додатку, є критичним для забезпечення високої продуктивності та надійності системи. Оскільки сервер має управляти великими обсягами даних та забезпечувати швидкий відгук на запити користувачів, необхідна значна кількість оперативної пам'яті. Оптимальний обсяг оперативної пам'яті залежатиме від кількості кешованих даних, але для більшості сучасних веб-додатків рекомендується від 64 ГБ до 128 ГБ RAM або більше, особливо якщо сервер буде обслуговувати велику кількість одночасних з'єднань або виконувати обчислення з великим обсягом даних.

Важливо вибрати оперативну пам'ять із високою швидкістю та низькою латентністю для підтримки швидкого доступу до даних, що зменшить час очікування користувачів і підвищить загальну продуктивність системи. Також, враховуючи потребу в надійності та безперебійній роботі сервера, розгляд можливості використання ECC (Error-Correcting Code) [22] пам'яті може бути розумним рішенням. ECC пам'ять здатна виявляти та виправляти помилки, що можуть виникнути в процесі роботи, тим самим забезпечуючи додатковий рівень захисту даних та стабільності системи.

Окрім обсягу та типу оперативної пам'яті, необхідно також враховувати можливості масштабування системи. Вибір серверної платформи, яка дозволяє легко додавати додаткову оперативну пам'ять у майбутньому, може забезпечити гнучкість для адаптації до зростаючих вимог додатку без необхідності повного оновлення обладнання. Такий підхід дозволить оптимізувати витрати та забезпечити, щоб система залишалася конкурентоспроможною та відповідала потребам користувачів у довгостроковій перспективі.

Hynix DDR4-2666 128Gb (4x32Gb) ECC Registered Memory Kit [23], можна переглянути на рисунку 2.3, видається раціональним вибором для такого сценарію

використання. З надійними характеристиками ECC (Error-Correcting Code), ця пам'ять забезпечує високий рівень захисту від можливих помилок, що є критичним у важливих застосунках. Обсяг в 128 ГБ і форм-фактор 4*32 ГБ дозволяють забезпечити велику кількість оперативної пам'яті, що може бути використана для ефективного керування даними та забезпечення швидкості роботи під час обробки великого потоку інформації.

Рисунок 2.3 – Зовнішній вигляд Hynix DDR4-2666 128Gb (4x32Gb)

DDR4-2666 є стандартом, який забезпечує високу швидкість передачі даних, а ECC та Registered підтримують стабільну та надійну роботу системи в умовах великого навантаження. Вибір такої пам'яті дозволяє оптимізувати продуктивність та забезпечувати надійну роботу сервера при роботі з великим обсягом даних.

Також не менш важливим є вибір жорсткого диска (SSD) для сервера, який буде використовуватися для хостингу додатку. Серверний SSD [24] повинен мати

високу швидкість передачі даних, низьку латентність та велику місткість для забезпечення ефективного зберігання та швидкого доступу до даних.

Одним з можливих варіантів є використання NVMe SSD дисків [25], які забезпечують найвищий рівень продуктивності завдяки використанню передової технології підключення PCI Express. NVMe SSD диски мають значно швидший рівень читання та запису даних порівняно з традиційними SATA SSD або HDD, що дозволяє зменшити час завантаження додатків та підвищити продуктивність сервера, але й в свою чергу мають значно менший ресурс витривалості.

Також, важливо обрати SSD диск із великою ресурсом витривалості та підтримкою технологій TRIM та S.M.A.R.T., що забезпечить довговічну та надійну роботу пристрою. Оптимальний вибір SSD диску також може включати аналіз відгуків користувачів та експертних оглядів, щоб забезпечити оптимальне співвідношення між продуктивністю, надійністю та вартістю пристрою.

На рисунку 2.4 зображено SSD HP Enterprise EO0400JEFPE (6765290-002, 802907-001) з об'ємом 400 GB [26]. Це високоякісний продукт від відомого виробника з надійною репутацією, що забезпечує стабільну роботу сервера та високу швидкість обробки даних.

Рисунок 2.4 – Зовнішній вигляд накопичувача SSD HP Enterprise SSD EO0400JEFPE

Використання такого SSD дозволяє значно прискорити зчитування та запис інформації, що покращує загальну продуктивність системи та забезпечує плавну і безперебійну роботу додатків. Обраний SSD має обсяг 400 ГБ, що забезпечить достатньо простору для зберігання додатку та його даних, а також дозволить вам зосередитися на розширенні додатку, не переймаючись про нестачу місця на накопичувачі. Такий вибір дозволить забезпечити ефективну роботу додатку та задовольнити потреби користувачів у швидкій та надійній роботі.

Перейдемо до вибору материнської плати для сервера, що буде використовуватися для хостингу додатку, важливо враховувати кілька ключових факторів для забезпечення оптимальної продуктивності, надійності та масштабованості системи.

По-перше, слід уважно розглянути сумісність материнської плати з іншими компонентами системи, такими як процесор, оперативна пам'ять та SSD диск. Наступним важливим аспектом є наявність достатньої кількості роз'ємів розширення для підключення додаткових пристроїв, таких як сховища даних, мережеві адаптери та інші. Також, материнська плата повинна підтримувати сучасні інтерфейси для швидкого обміну даними, такі як M.2 та PCIe. Крім того, важливим аспектом є надійність материнської плати та її підтримка від виробника.

Вибір материнської плати має велике значення для успішного функціонування сервера, тому важливо ретельно аналізувати всі аспекти її сумісності, функціональності та надійності перед придбанням. На рисунку 2.5 зображена материнська плата ASUS P10S-E WS [27], яка є чудовим варіантом для серверних застосувань, таких як хостинг додатків. Вона відрізняється високою надійністю та сумісністю з процесорами Intel Xeon E3-1200 v5, що забезпечує стабільну і ефективну роботу сервера.

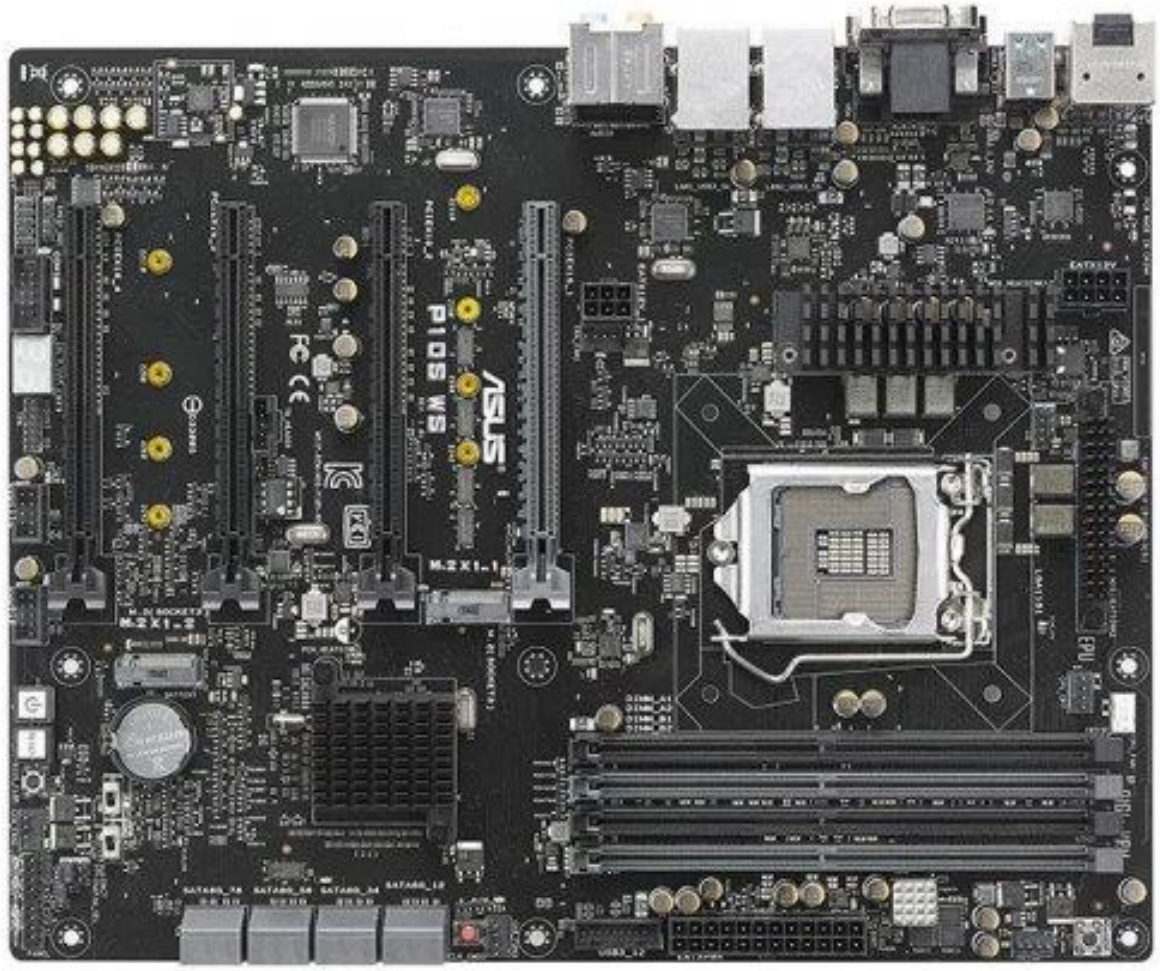


Рисунок 2.5 – Зовнішній вигляд материнської плати ASUS P10S-E WS

Основні переваги ASUS P10S-E WS включають можливості розширення та підтримку великої кількості оперативної пам'яті, що ідеально відповідає потребам у зберіганні багато оперативної пам'яті для кешування даних. Завдяки підтримці ECC пам'яті, ASUS P10S-E WS забезпечує підвищений рівень надійності і захисту даних, що особливо важливо у сфері хостингу додатків.

Також важливою особливістю цієї материнської плати є її висока масштабованість і підтримка розширених інтерфейсів для підключення пристроїв зберігання даних, мережевих контролерів та інших додаткових компонентів, що дозволяє зручно розширювати функціональні можливості сервера в майбутньому.

Загалом, материнська плата ASUS P10S-E WS є відмінним вибором для серверної платформи з урахуванням її можливостей розширення, підтримки ECC

пам'яті та високої надійності, що робить її ідеальним варіантом для хостингу додатків та підтримує велику кількість оперативної пам'яті для кешування даних.

2.3 Проєктування архітектури системи

Архітектура клієнтської частини, побудованої на основі Next.js для веб-додатку, є важливим компонентом програмно-технічної системи обробки, зберігання та передачі відеоконтенту. Next.js, як сучасний фреймворк для розробки React-додатків, надає можливість створення серверно-рендерингу сторінок [28], що значно підвищує продуктивність і SEO-оптимізацію веб-додатків [29]. Архітектура Next.js дозволяє ефективно організувати роботу з відеоконтентом, забезпечуючи швидке завантаження, обробку та передачу даних.

Основу архітектури клієнтської частини складають декілька ключових компонентів, кожен з яких виконує специфічні функції. Одним із важливих елементів є компонент для серверно-рендерингу сторінок (Server-Side Rendering, SSR) [28]. Цей компонент відповідає за попереднє рендерування сторінок на сервері перед тим, як вони будуть відправлені клієнту. SSR забезпечує також підвищену безпеку, оскільки більшість обчислень відбувається на сервері, а не на клієнтському пристрої. Основні механізми роботи SSR, зображено на рисунку 2.6.

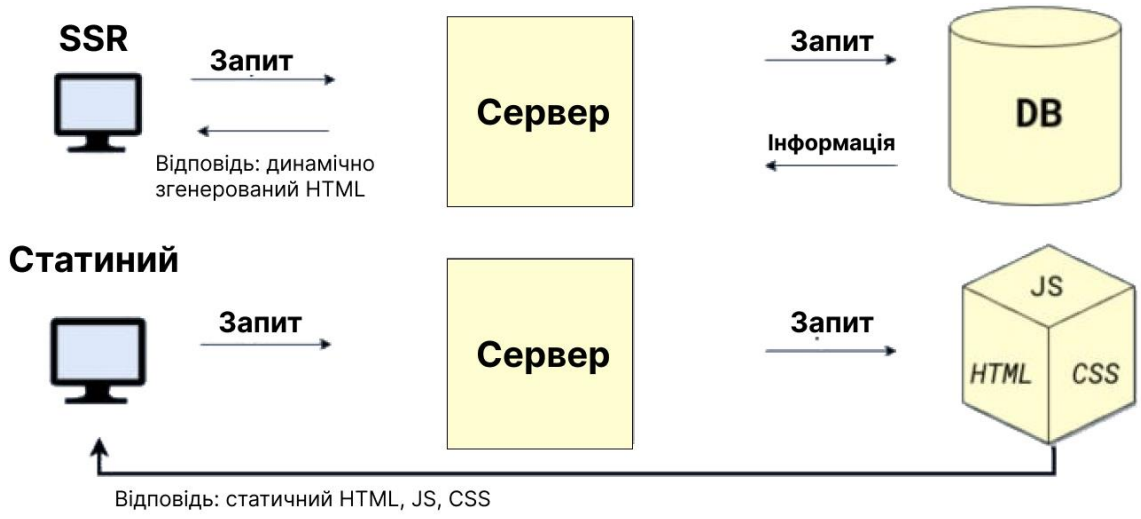


Рисунок 2.6 – Принцип роботи серверного рендерингу сторінок HTML

Наступним важливим компонентом є система маршрутизації, яка забезпечується за допомогою вбудованого роутера Next.js. Цей компонент відповідає за навігацію користувача між різними сторінками додатка. Завдяки маршрутизатору можна динамічно завантажувати різні частини додатка, що дозволяє оптимізувати використання ресурсів та зменшити навантаження на сервер. Система маршрутизації також підтримує функціонал автоматичної генерації маршруту на основі структури файлів, що спрощує процес розробки та підтримки коду. Next.js пропонує інтегрований підхід до організації маршрутизації через App Router [30]. App Router дозволяє легко визначати маршрути додатка за допомогою файлової системи. Кожен файл з назвою «index.ts» відповідає сторінці, що спрощує структуру проекту та полегшує навігацію.

Однак, App Router працює чудово для простих і середньо складних додатків, де збереження стану не є критичною вимогою. Проте, для більш складних додатків з вимогами до персистентного збереження даних, можуть виникнути значні труднощі.

Для реалізації клієнтської частини проекту буде використано Pages Router [31], який надає більше гнучкості та кращу інтеграцію з бібліотеками для збереження стану, такими як Redux Persist [32]. Pages Router у Next.js побудований на основі файлової системи, де кожен файл або директорія у директорії "pages" автоматично стає маршрутом для додатка. Цей підхід не лише спрощує створення нових сторінок, але й забезпечує чітку та організовану структуру проекту. Така структура дозволяє розробникам легко розуміти і управляти маршрутизацією у проекті без необхідності писати додатковий код для визначення маршрутів. Важливою перевагою Pages Router є його сумісність із популярними бібліотеками для управління станом, такими як Redux [33] і Redux Persist [32]. Архітектура Pages Router [31] дозволяє легко інтегрувати Redux Persist [32], що забезпечує збереження стану додатка у локальному сховищі браузера. Це значно спрощує зміну станів кожного компонента окремо, забезпечуючи

ефективне та зручне управління станом додатка. Відмінності та переваги цього підходу добре ілюстровані на рисунку 2.6.

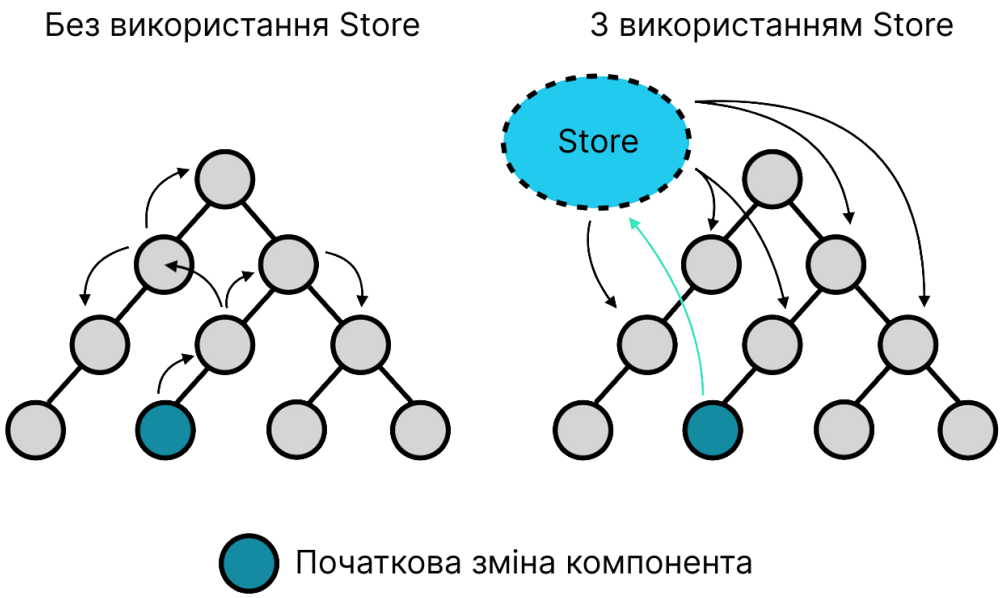


Рисунок 2.6 – Різниця зміни стану компонентів з та без використанням Persist Store

Крім того, використання Redux Persist Store дозволяє зменшити навантаження на сервери, оскільки дані користувача зберігаються локально і не потребують постійної синхронізації з сервером. При кожному завантаженні додатка стан автоматично відновлюється з цього сховища, що забезпечує безперервність користувацького досвіду. Це підвищує продуктивність додатка і зменшує час завантаження сторінок, що є важливим фактором для відеоконтенту, де швидкий доступ до даних користувача має вирішальне значення.

2.4 Проєктування інтерфейсу користувача

Розробка інтерфейсу користувача для програмно-технічної системи є ключовою складовою процесу розробки. Вона має свої унікальні особливості, але загальним є створення зручного та інтуїтивно зрозумілого інтерфейсу, який відповідає потребам користувачів.

Також необхідно врахувати можливость вибору білої та темної теми (рисунок 2.7) для додатків [34], що дозволить користувачам налаштувати інтерфейс відповідно до їхніх вподобань і забезпечить комфортне користування в будь-який час доби.

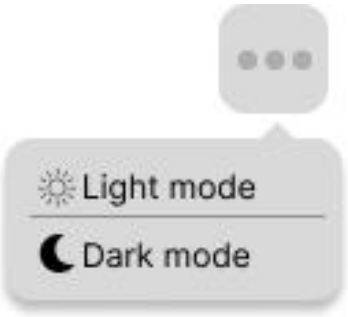


Рисунок 2.7 – Дизайн перемикача теми додатку

На рисунку 2.8 зображені всі форми, які використовуються на сторінці авторизації. Дизайн цієї сторінки спрямований на забезпечення інтуїтивного та зручного користувацького досвіду. Сторінки авторизації містить три основні форми: вхід, реєстрація та відновлення паролю.

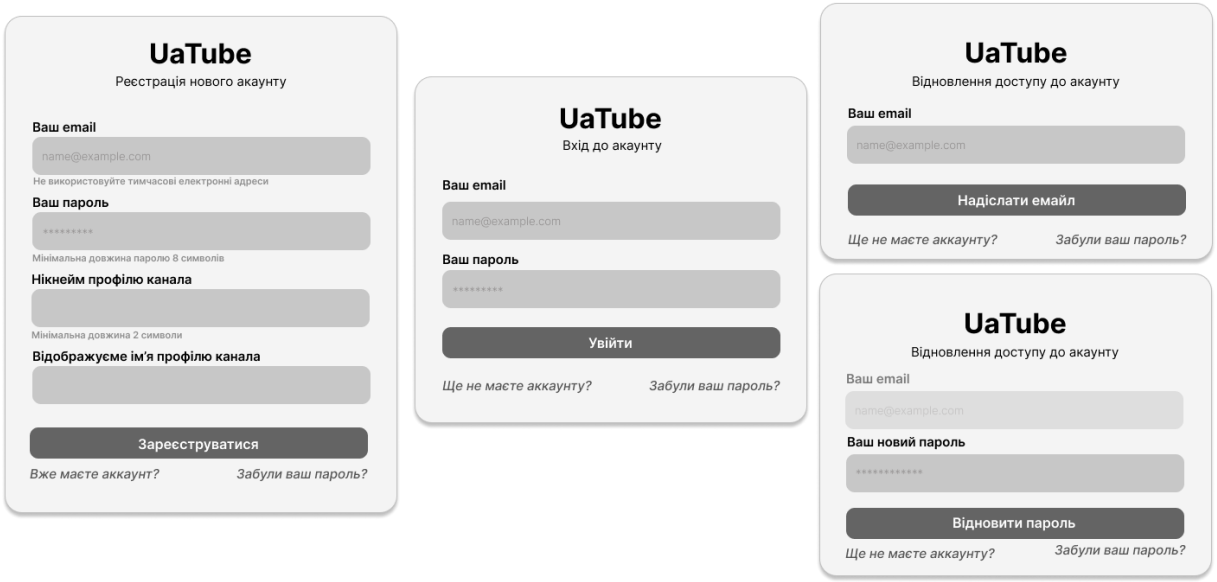


Рисунок 2.8 – Зображення макету сторінки авторизації

Форма входу включає два поля: для введення електронної пошти та пароля. Вона розташована в центрі екрану і має простий, але елегантний дизайн з чіткими полями для введення даних та великою кнопкою "Вхід", яка забезпечує зручність використання. Під полями входу розміщені посилання на форми реєстрації та відновлення паролю, що дозволяє легко перемикатися між ними.

Форма реєстрації складається з чотирьох полів: електронна пошта, пароль, назва каналу, нікнейм та вибірково фото профілю. Ця форма також розташована в центрі екрану та оформлена у тому ж стилі, що й форма входу, для збереження єдиного стилю. Поля для введення мають зрозумілі підписи, що допомагає користувачам швидко заповнити необхідну інформацію. Кнопка "Реєстрація" виділяється на фоні інших елементів, привертаючи увагу користувачів.

Форма відновлення паролю включає два поля, які динамічно з’являються, в залежності від поточного кроку: електронна пошта та новий пароль. Вона аналогічно розташована в центрі екрану і має схожий дизайн з іншими формами. Після введення необхідних даних користувач може натиснути кнопку "Відновити пароль".

Загальний дизайн сторінки авторизації є мінімалістичним і сучасним, з акцентом на зручність та функціональність. Використання єдиної кольорової схеми та стильових елементів створює гармонійний вигляд, що сприяє позитивному користувацькому досвіду. Всі елементи дизайну розташовані логічно і зрозуміло, що забезпечує легкість навігації та швидкий доступ до необхідних функцій.

Також розглянемо декілька ключових сторінок, основними з яких є головна сторінка з безліччю відео та сторінка одного відео. Дизайн додатку розроблений з урахуванням сучасних тенденцій та забезпечує зручність користування.

На головній сторінці, яка зображена на рисунку 2.9, користувачі побачать відео в форматі сітки, що дозволяє швидко переглядати доступний контент. Кожне відео буде представлено у вигляді мініатюри з назвою, кількістю переглядів та короткою інформацією про автора. Над сіткою відео розташовані

вкладки для швидкого доступу до різних категорій контенту, таких як трендові відео, рекомендовані, нові надходження тощо. Бічна панель містить посилання на інші розділи сайту, такі як підписки, історія переглядів, список відтворення та налаштування профілю користувача. Верхня панель містить логотип платформи, поле для пошуку, іконки для сповіщень, повідомлень та профілю користувача.

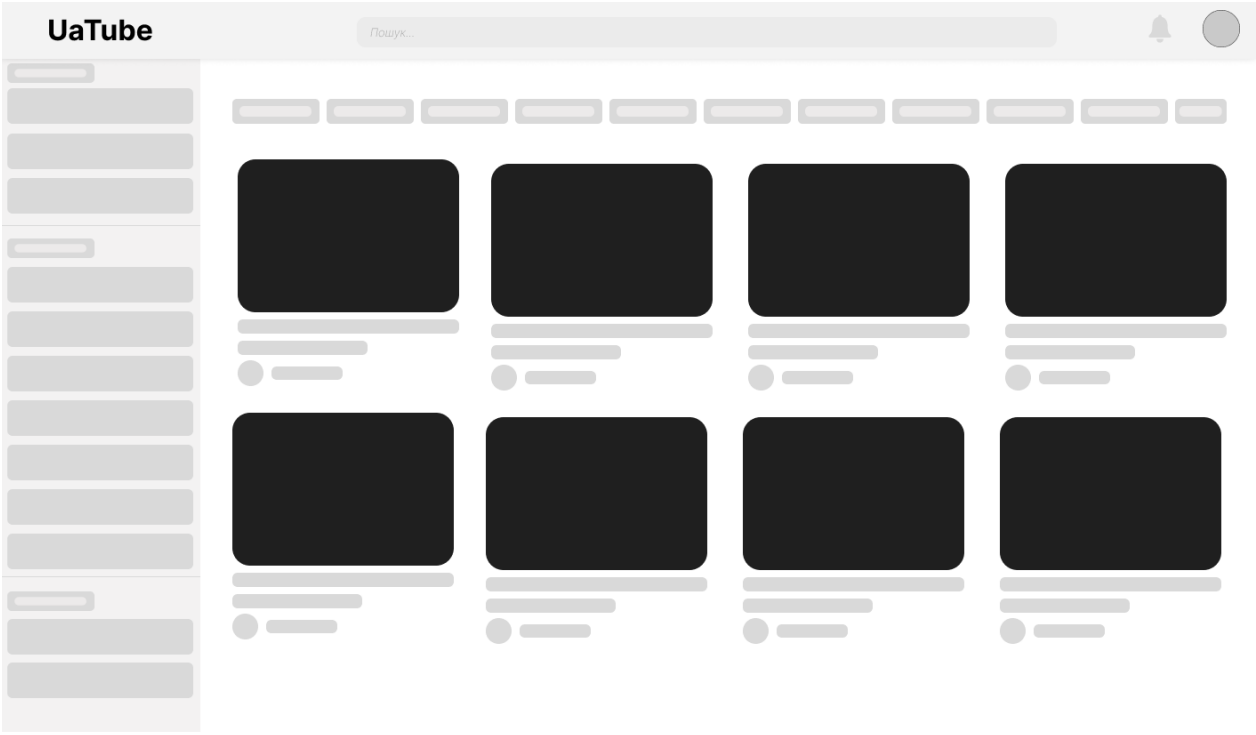


Рисунок 2.9 – Зображення дизайну головної сторінки веб-додатку

На рисунку 2.10 зображена сторінка одного відео, яка включає великий відеоплеєр, який займає центральне місце. Під відео розташовані елементи управління: кнопки "подобається" та "не подобається", кнопка підписки на канал, лічильник переглядів, дата публікації та розділ з коментарями. Праворуч від відеоплеєра буде відображено список рекомендованих відео, які користувач може переглянути далі. Це забезпечує зручну навігацію та допомагає користувачам знаходити більше цікавого контенту.

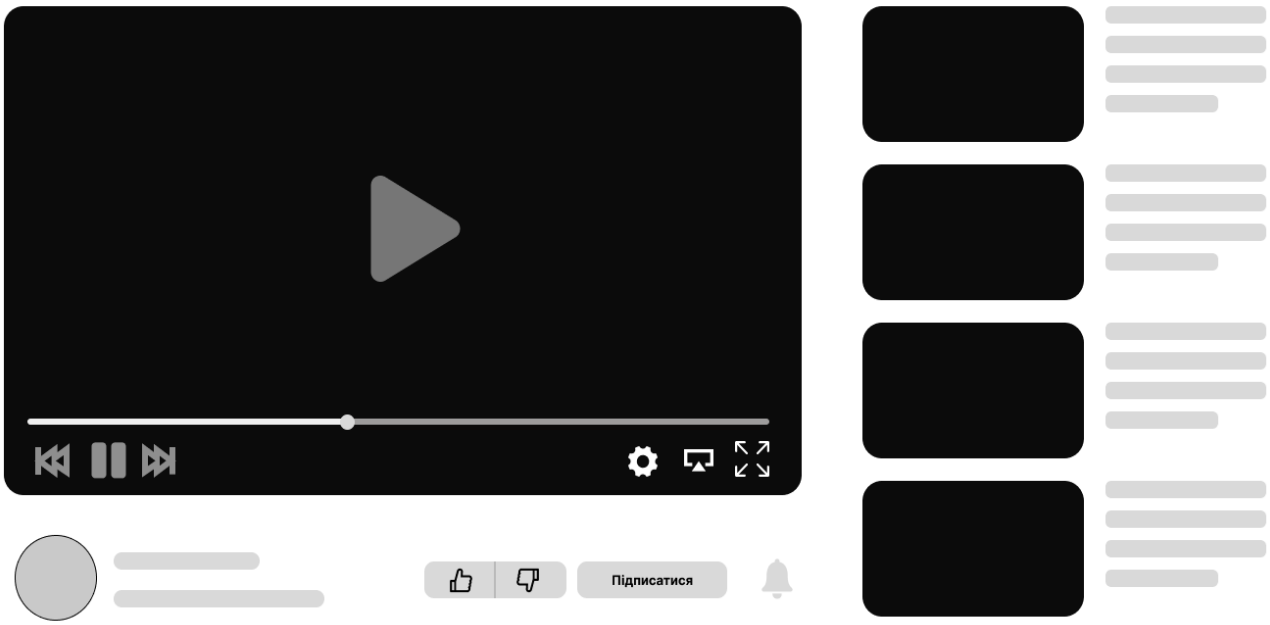


Рисунок 2.10 – Зображення дизайну сторінки перегляду відео у веб-додатку

2.5 Забезпечення захисту інформації

Забезпечення захисту інформації є критично важливим аспектом будь-якої сучасної веб-системи, особливо коли мова йде про клієнтську частину, де відбувається безпосередня взаємодія з користувачами. Аналіз загроз безпеки включає виявлення можливих векторів атак, таких як підробка міжсайтових запитів (CSRF), крадіжка даних сесій, злом автентифікаційних механізмів та інші види кіберзагроз.

Основним методом захисту від CSRF-атак [35] буде впровадження токенів CSRF [36]. Ці токени генеруються на сервері та додаються до кожної форми або запиту, який користувач надсилає до сервера. Коли сервер отримує запит, він перевіряє наявність і коректність CSRF-токена, щоб упевнитися, що запит походить від автентичного користувача і не є результатом злочинної маніпуляції.

Для забезпечення безпечної автентифікації та авторизації користувачів використовуються JWT-токени (JSON Web Tokens) [37]. Після успішного входу користувача в систему, сервер видає йому JWT-токен, який містить інформацію про користувача та його права доступу. Основна структура даного токена

зображена на рисунку 2.11. Цей токен підписується за допомогою секретного ключа, що дозволяє серверу верифікувати його справжність при кожному запиті користувача. Використання JWT-токенів забезпечує безпечну передачу інформації між клієнтом і сервером, оскільки токен можна легко перевірити на підробку та зміну. Такий механізм автентифікації та авторизації забезпечує високий рівень безпеки, захищаючи систему від несанкціонованого доступу та підробки даних.

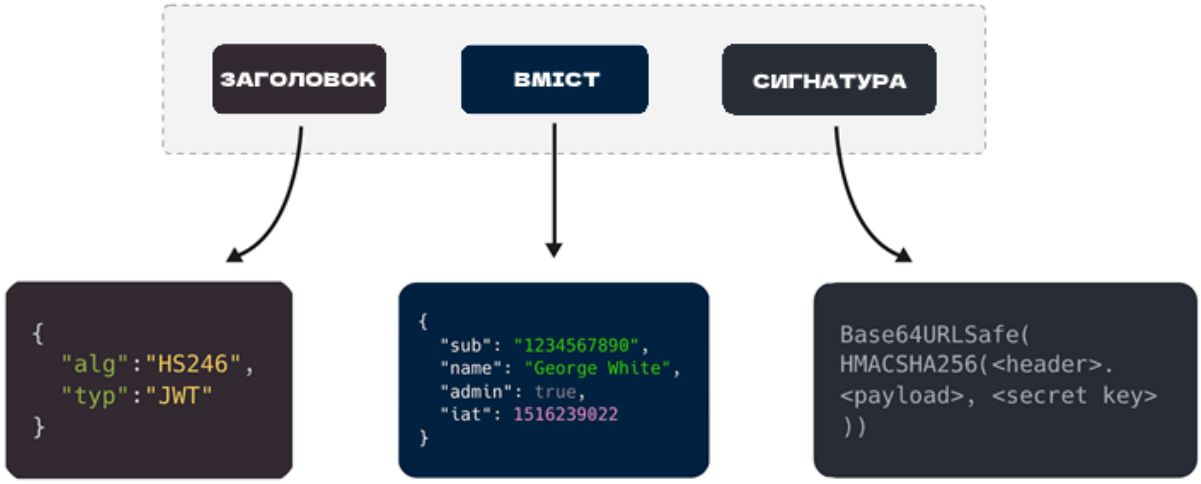


Рисунок 2.11 – Механізм роботи JWT токену

Додатково до цього, система повинна використовувати HTTPS [38] для шифрування всього трафіку між клієнтом і сервером, що запобігає можливим атакам типу "людина посередині" (MITM) [39]. Регулярне оновлення програмного забезпечення, проведення безпекових аудитів та тестування на проникнення також є важливими елементами комплексної стратегії захисту інформації. Ці заходи допоможуть виявляти та виправляти вразливості, підтримуючи високий рівень безпеки для користувачів системи.

Також одним із важливих аспектів архітектури клієнтської частини, що забезпечує додатковий захист відеоконтенту, є використання HLS (HTTP Live Streaming) файлів [40]. HLS є популярним протоколом для потокової передачі мультимедіа, розробленим компанією Apple [41], який забезпечує не лише

ефективне відтворення відео на різних пристроях, але й підвищену безпеку контенту.

Основною перевагою HLS є те, що він розбиває відеопотік на невеликі сегменти, які передаються клієнтському пристрою послідовно. Кожен сегмент представляє собою короткий відеофайл, що зберігається на сервері і передається користувачеві за запитом. Такий підхід дозволяє забезпечити адаптивну потокову передачу, коли якість відео автоматично змінюється залежно від швидкості інтернет-з'єднання користувача. Однак, окрім поліпшення користувацького досвіду, HLS також надає можливості для додаткового захисту відеоконтенту.

Захист відео за допомогою HLS досягається шляхом шифрування сегментів відео. HLS підтримує AES-128 шифрування [42], що дозволяє зашифрувати кожен сегмент відео, перед тим як він буде переданий клієнту. Для цього використовується спеціальний ключ шифрування, який зберігається на сервері і передається клієнту через захищений канал. Це гарантує, що лише авторизовані користувачі можуть отримати доступ до відеоконтенту, оскільки для його розшифрування необхідно отримати відповідний ключ.

Загалом, комплексний підхід до захисту інформації, що включає використання CSRF-токенів, JWT-токенів, шифрування трафіку за допомогою HLS та регулярний моніторинг, дозволить забезпечити високий рівень безпеки для клієнтської частини системи.

2.4 Висновки

У розділі 2 проаналізовано апаратні та програмні підсистеми для успішної реалізації проекту. Операційні системи обрані з поєднанням Windows для розробки та Linux для продуктивного середовища, що забезпечує гнучкість і стабільність. Мови програмування TypeScript і JavaScript вибрані для створення динамічної клієнтської частини.

Розглянуто бібліотеки та фреймворки, зокрема Next.js, що забезпечують високу продуктивність у розробці веб-додатків. Вибір апаратного забезпечення, включаючи серверні процесори, графічні процесори, оперативну пам'ять та SSD-диски, орієнтований на максимальну ефективність і надійність для безперебійного хостингу та швидкого оброблення даних.

Забезпечення захисту інформації здійснюється за допомогою CSRF- і JWT-токенів, що підвищує рівень безпеки системи, запобігаючи атакам та несанкціонованому доступу.

3 РЕАЛІЗАЦІЯ ПРОГРАМНО-ТЕХНІЧНОЇ СИСТЕМИ

3.1 Створення та налаштування проєкту

На початку створення програмно-технічної системи необхідно встановити та налаштувати основні інструменти та бібліотеки, що будуть використовуватися в процесі розробки клієнтської частини. Використовуючи Next.js, дозволить отримати переваги серверного рендерингу та оптимізації продуктивності, що є важливими для сучасних систем.

Першим кроком є створення нового Next.js проєкту, при створенні проєкту одразу можна обрати встановлення та ініціалізацію Tailwind CSS [44] та Typescript [14]. Для цього потрібно відкрити термінал і виконати команду, яку наведено нижче:

npx create-next-app@latest uatube_client

Результат успішного виконання даної команди з вибором автоматичного встановлення Tailwind CSS та Typescript можна побачити на рисунку 3.1.

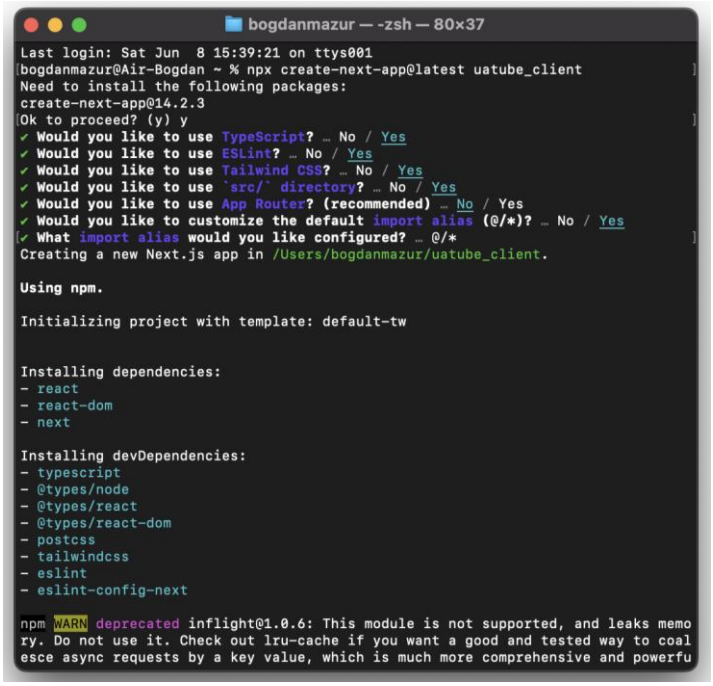


Рисунок 3.1 – Успішне виконання команди створення проєкту Next.JS

Після цього виконується перехід до створеної директорії проєкту, використовуючи команду:

cd uatube_client

Наступним кроком є встановлення бібліотек для користувацького інтерфейсу. Встановлюємо та ініціалізуємо Shadcn UI [45], який забезпечить зручні та гнучкі компоненти:

npx shadcn-ui@latest init

Результатом виконання та вибору необхідних даних зображено на рисунку 3.2.

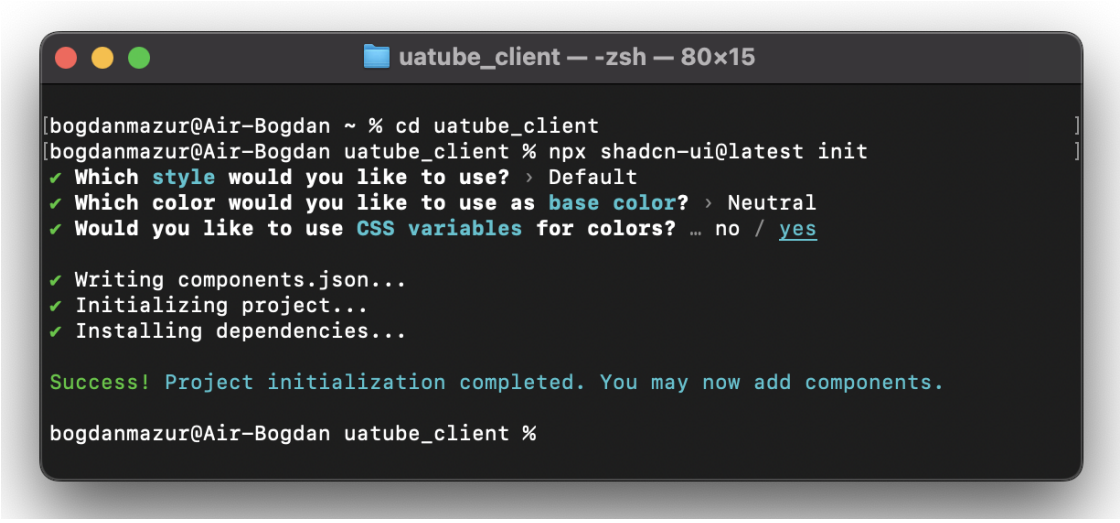


Рисунок 3.2 – Успішне виконання команди ініціалізації бібліотеки Shadcn UI

Не менш необхідним є встановлення бібліотек для локалізації додатку, для цього потрібно буде встановити декілька бібліотек, які всі базуються на бібліотеці i18n [46]. Встановимо їх за допомогою даної команди:

npm install next-i18next i18next

А також оскільки Next.Js базується на React.Js додатково буде потрібно встановити ще один NPM пакет [43] в залежності для розробки, за допомогою команди, яка знаходится нижче:

npm install -D react-18next

Після успішного встановлення необхідних пакетів для локалізації, потрібно створити новий файл у директорії проєкту під назвою next-i18next.config.js. Цей файл дозволить визначити всі локалізації, доступні у проєкті, а також встановити локалізацію за замовчуванням, яку буде призначено українською мовою. Після створення цього файлу, його вміст необхідно змінити відповідно до зображення на рисунку 3.3. Це дозволить додати такі мови, як українська, французька, німецька, англійська та інші популярні мови ЄС, а також автоматично перезавантажувати проєкт під час зміни локалей.

```
module.exports = {
  i18n: {
      defaultLocale: 'uk',
      locales: ['uk', 'fr', 'de', 'tr', 'pl', 'es', 'it', 'en'],
      localeDetection: false
  },
  reloadOnPrerender: process.env.NODE_ENV === 'development'
}
```

Рисунок 3.3 – Новий вміст файлу next-i18next.config.js

Також необхідно встановити бібліотеки для роботи з формами такі як Zod, який необхідний для валідації та React Hook Form, що надає потужні можливості для створення та валідації форм, це можна зробити за допомогою команди npm install react-hook-form zod.

У результаті успішного налаштування проєкту, проєкт повністю готовий до розробки, завдяки використанню сучасних інструментів і бібліотек, таких як Next.js, TypeScript, Tailwind CSS, Shadcn UI, Radix UI, i18n, React Hook Form та Zod надається можливість швидко та ефективно розробляти клієнтську частину проєкту.

3.2 Розробка архітектури

Для правильної роботи та масштабованості системи потрібно організувати правильну структу проєкту. Загальну структуру проєкту зображено на рисунку 3.4.

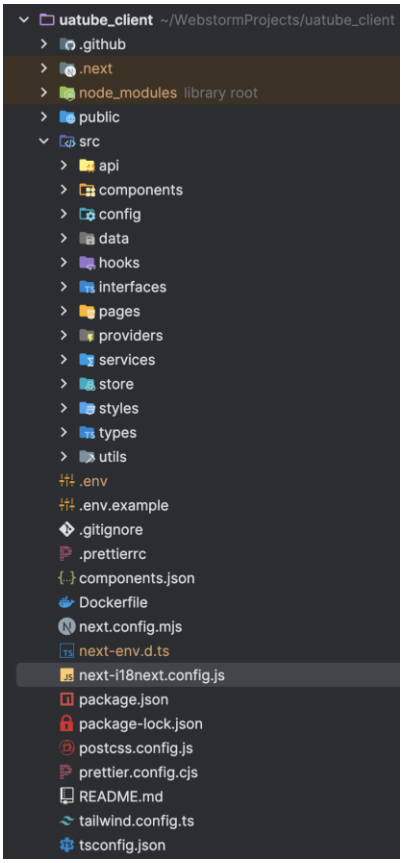


Рисунок 3.4 – Структура розробленого додатку на Next.Js

Проєкт містить кілька основних директорій і файлів. Папка .github містить налаштування GitHub Actions та інші файли, пов'язані з GitHub, що автоматизують процеси CI/CD [47]. Папка .next генерується автоматично під час розробки Next.js і містить зібрані файли та інші артефакти, потрібні для рендерингу сторінок. node_modules - це папка, яка містить всі встановлені npm-пакети, необхідні для роботи проєкту.

Папка public – тут розміщуються статичні файли, які можуть бути напряму доступні з кореня сайту, наприклад, зображення, шрифти або інші ресурси.

Основна папка з вихідним кодом проєкту – src – має кілька підпапок, що організують код за різними категоріями. Папка api зберігає файли для взаємодії з API. Вона містить функції для викликів до бекенду та обробки відповідей. Папка components містить компоненти інтерфейсу користувача, які використані в різних частинах додатку, наприклад, кнопки, модальні вікна, навігаційні панелі тощо. Папка config зберігає конфігураційні файли, які визначають поведінку додатку. Папка data зберігає дані, які можуть бути використані в додатку, наприклад, файли з демо-даними або константами.

Папка hooks містить власні React-хуки, які повторно використовуються в різних компонентах. Папка interfaces зберігає TypeScript інтерфейси та типи, які визначають структуру даних, що використовуються в проєкті. Папка pages містить сторінки додатку, кожен файл у цій папці відповідає окремій сторінці, яка буде доступна за певним URL, а також файл _document.tsx, який використовується для задання структури HTML коду, та app.tsx в якому визначається структура додатку. Папка providers зберігає провайдерів контексту, які можуть надавати глобальний стан або інші функціональні можливості через React Persist Store. Папка services містить сервіси, які виконують бізнес-логіку або взаємодіють з API. Папка store призначена для управління глобальним станом додатку за допомогою бібліотеки Redux. Папка styles містить стилі. Тут зберігаються глобальні CSS-файли, конфігурація для Tailwind CSS та Shadcn UI. Папка types зберігає додаткові TypeScript типи, які використані в проєкті. Папка utils містить утиліти, що містять допоміжні функції.

Крім того, існують файли .env та .env.example, які зберігають змінні середовища, де .env.example використовується як приклад для конфігурації. Файл .gitignore визначає, які файли та папки ігнорувати при коміті в репозиторій. Файл .prettierrc конфігурує Prettier, інструмент для автоматичного форматування коду. Файл components.json визначає конфігурацію для компонентів Shadcn UI. Dockerfile налаштовує Docker-контейнер, що дозволяє запускати додаток у стандартизованому середовищі. Файл next.config.mjs конфігурує Next.js,

визначаючи поведінку фреймворку. Файл next-i18next.config.js конфігурує бібліотеку i18n, налаштовуючи локалізацію в додатку. Файли package.json містить метадані проєкту та список залежностей. Файл postcss.config.js конфігурує PostCSS, інструмент для трансформації CSS з плагінами. Файл README.md містить документацію проєкту. Файл tailwind.config.ts конфігурує Tailwind CSS. Файл tsconfig.json конфігурує TypeScript, визначаючи налаштування компілятора.

Правильна структура додатку є необхідною для коректно налаштованої системи. Вона забезпечує чітку організацію коду, що полегшує його підтримку та розвиток. Кожна папка має своє призначення, що дозволяє легко знайти та змінити необхідні частини коду та забезпечити масштабованість системи.

Не менш важливим аспектом успішного проєкту, окрім правильної структури додатку, є компоненти, що дозволяють ефективно розділяти код для багаторазового використання. Використання компонентів забезпечує модульність, повторне використання коду, легкість у тестуванні та обслуговуванні. Компоненти дозволяють розробникам створювати ізольовані блоки функціональності, що можуть бути з легкістю інтегровані в різні частини додатку. Це зменшує дублювання коду і сприяє створенню більш чистої та підтримуваної архітектури додатку. Нижче наведена таблиця 3.1, яка описує основні компоненти, що використовуються в проєкті:

Таблиця 3.1 – Основні компоненти системи

Компонент	Опис
1	2
VideoList	Відображає список відео, включаючи прев'ю, назву та іншу важливу інформацію.
ShareVideoModal	Модальне вікно для поширення відео.
AppHead	Компонент для налаштування мета-тегів і заголовків сторінки.

Продовження таблиці 3.1. – Основні компоненти системи

1	2
VideoCommentsList	Відображає список коментарів до конкретного відео.
PlaylistList	Відображає список плейлістів, доступних на каналі або у користувача.
VideoCategoriesPills	Компонент для відображення категорій відео у вигляді пігулок.
SignUpForm	Форма реєстрації нового користувача.
SignInForm	Форма для входу в систему зареєстрованого користувача.
RecoveryPassForm	Форма для відновлення забутого пароля користувача.
LoginLayout	Макет сторінок авторизації, включаючи логін, реєстрацію та відновлення пароля.
HomeLayout	Макет головної сторінки з популярними відео.
StudioLayout	Макет студії для управління контентом користувача.
PlaylistsModal	Модальне вікно для додання відео в плейліст.
VideoPlayer	Відтворювач відео, що дозволяє програвати відео контент з можливістю контролю.
PlaylistCreateForm	Форма для створення нового плейліста.
AboutChannel	Компонент для відображення загальної інформації про канал користувача.

Кінець таблиці 3.1. – Основні компоненти системи

1	2
AboutChannelModal	Модальне вікно для перегляду детальної інформації про канал користувача.
VideosTable	Таблиця для відображення відео з пагінацією.
VideoEditTab	Вкладка в кабінеті користувача, для редагування даних про відео.
VideoCreateModal	Модальне вікно створення початкової інформації про відео.
VideoUploadModal	Модальне вікно завантаження відео на сервер та подальшої обробки.
HistoryContent	Компонент для відображення історії.

Важливо відзначити, що представлені компоненти не є ізольованими елементами, а взаємодіють між собою, формуючи єдину екосистему. Наприклад, компонент VideoList може використовувати VideoCard для вивід інформації про відео, а ShareVideoModal може інтегруватися з VideoList для поширення вибраного відео. Така модульна структура забезпечує гнучкість та масштабованість системи, дозволяючи легко додавати нові функції та змінювати існуючі без значного впливу на інші частини додатку.

3.3 Розробка інтерфейсу

Весь інтерфейс додатку було реалізовано за допомогою Tailwind CSS, що забезпечує ефективний та зручний спосіб стилізації компонентів. Tailwind CSS дозволяє застосовувати стилі безпосередньо в HTML-розмітці, використовуючи класові атрибути, що спрощує процес створення і підтримки коду.

До приклада розглянемо компонент HomeHeaderLogo [54]. Цей компонент відповідає за відображення логотипу в шапці головної сторінки, включаючи інтерактивні елементи для користувачів, що залоговані. Загальний код компонента можна глянути нижче:

Цей компонент повертає div з класами Tailwind CSS для створення адаптивної та стилізованої структури. Якщо пропс «hidden» дорівнює «true», елемент приховується, інакше він відображається як flex-контейнер з відступом між елементами (gap-4) і вертикальним вирівнюванням (items-center).

Для залогованого користувача відображається кнопка з класами для округлення кутів, зміни фону при наведенні, вирівнювання та відступів, іконка якої змінюється залежно від стану бокової панелі.

Компонент Link від Next.js використовується для переходу на головну сторінку і містить класи для позиціонування, відступів та вирівнювання. Всередині посилання розміщено зображення логотипу з висотою (h-6) і текст назви системи з напівжирним шрифтом (font-semibold). Локаль користувача показується за допомогою класів для абсолютного позиціонування, великих літер, кольору тексту, розміру шрифту та позиціонування зверху праворуч.

Також для створення інтерфейсу було використано бібліотеку ShadcnUi, яка забезпечує набір компонентів, що легко інтегруються та дозволяють гнучко налаштовувати їхній вигляд і поведінку. Компоненти, що використовуються в проєкті, включають широкий спектр елементів інтерфейсу, таких як:

1. Avatar - для відображення аватарів користувачів.
2. Badge - для відображення бейджів або міток.
3. Button – для створення кнопок різних стилів і розмірів.
4. Card – для організації контенту в картках.
5. Checkbox – для створення чекбоксів.
6. Drawer – для реалізації бокових панелей, що можуть висуватись.
7. DropdownMenu – для створення випадаючих меню.
8. Form – для побудови форм введення даних.

9. Input – для створення полів введення.

10. Popover – для реалізації поповерхневих вікон.

11. Tabs – для організації кона.тенту в вкладках.

ShadcnUi надає можливість кастомізації компонентів через властивості, що дозволяють змінювати їхні стилі і поведінку. Наприклад, компонент Button може бути налаштований за допомогою параметрів для зміни кольору, розміру, іконок та інших властивостей, що робить його універсальним для використання в різних частинах додатку.

Однією з основних переваг використання компонентів ShadcnUi є їхня гнучкість і можливість розширення. Наприклад, компонент DropdownMenu може бути налаштований для підтримки багаторівневих меню, а компоненти Form і Input дозволяють створювати складні форми з валідацією і динамічними полями.

Крім цього, особливу увагу було приділено можливості зміни теми додатку. Всі компоненти ShadcnUi підтримують темізацію, що дозволяє легко змінювати кольорову схему і загальний стиль додатку відповідно до потреб користувача, а отже дозволяє легко адаптувати і власні компоненти під теми. Також щоб все працювало коректно потрібно встановити додатковий пакет під назвою Next Themes [49], виконаємо це за допомогою команди npm і next-themes.

Після успішного встановлення NPM пакету, потрібно створити ThemeProvider додавши настпуний код у файл компоненту:

```
import { ThemeProvider as NextThemesProvider } from 'next-themes'
import { type ThemeProviderProps } from 'next-themes/dist/types'
import { FC } from 'react'

const ThemeProvider: FC<ThemeProviderProps> = ({ children, ...props }) => {
  return (
   <NextThemesProvider
    attribute='class'
    defaultTheme='system'
    enableSystem
    {...props}
   >
    {children}
   </NextThemesProvider>
  )
}

export default ThemeProvider
```

Далі залишається лише додати функціонал для зміни теми, реалізуємо це за допомогою хука, який вже є в Next Themes, під назвою useTheme, код реалізації можна глянути нижче:

```
const { setTheme, theme } = useTheme()
```

Цей хук дозволяє змінювати тему на такі як світла, темна та системна, в свою чергу системна тема буде автоматично підбиратися в залежності від теми, яка вибрана в системі поточного користувача.

Загальний вигляд даного меню можна глянути на рисунку 3.5, який знаходиться нижче.

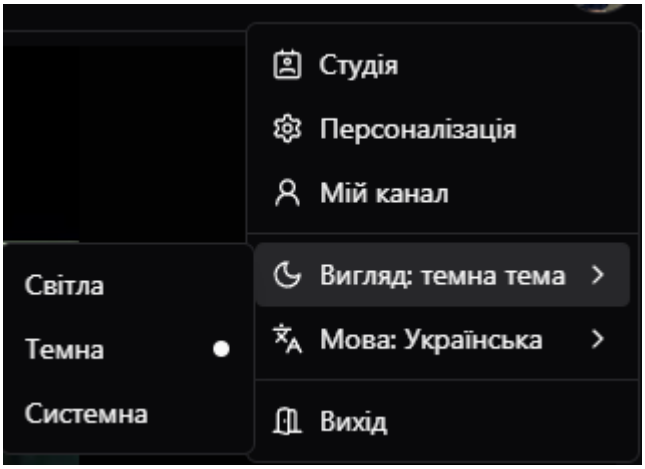


Рисунок 3.5 – Вигляд меню перемикання тем

Таким чином, використання ShadcnUi та Tailwind CSS в цьому проєкті не тільки спрощує процес розробки інтерфейсу, але і забезпечує високу гнучкість і можливість кастомізації, що є критично важливими для створення сучасного та зручного додатку.

3.4 Розробка функціоналу

На даному етапі необхідно реалізувати головний функціонал системи- це перегляд та завантаження відео з адаптивною зміною якості в залежності від пропускної швидкості мережі користувача, для цього встановимо бібліотеку HLS.Js [50], використавши команду npm i hls.js.

Після успішного встановлення бібліотеки, переходимо до реалізації відтворення відео, створивши новий компонент під назвою VideoPlayer (додаток Г). Після завантаження сторінки плеєр отримуватиме об'єкт відео, що містить поле masterPlaylist. Під час ініціалізації HLS.js це поле потрібно передати для завантаження плейліста. HLS.js автоматично аналізує маніфест і підключає відповідні рівні якості. Для цього використовуються методи loadSource та attachMedia. Після успішного завантаження маніфесту компонент оновлює свій стан, зберігаючи доступні рівні якості. Нижче наведено код ініціалізації:

```
useEffect(() => {
  const hlsInstance = new Hls({
   startLevel: -1,
   autoStartLoad: true,
   maxBufferLength: 60,
   maxBufferSize: 60 * 1000 * 1000,
   abrEwmaDefaultEstimate: 5 * 1000 * 1000
  })
  if (Hls.isSupported()) {
   hlsInstance.loadSource(video?.masterPlaylistUrl!)
   hlsInstance.attachMedia(videoRef.current!)

   hlsInstance.on(Hls.Events.MANIFEST_PARSED, () =>
    setVideoState(p => ({
     ...p,
     levels: hlsInstance.levels,
     selectedLevel: hlsInstance.currentLevel
    }))
   )

   setHls(hlsInstance)
  } else if (videoRef.current?.canPlayType('application/vnd.apple.mpegurl')) {
   videoRef.current.src = video?.masterPlaylistUrl!
  }

  return () => {
   if (hlsInstance) hlsInstance.destroy()
  }
}, [video])
```

Цей код використовує useEffect, щоб налаштувати відтворення відео за допомогою HLS (HTTP Live Streaming) при завантаженні компонента або при зміні властивості video.

Під час першого виклику useEffect створюється новий екземпляр HLS з заданими параметрами, такими як startLevel, autoStartLoad, maxBufferLength, maxBufferSize та abrEwmaDefaultEstimate. Ці параметри дозволяють налаштувати різні аспекти відтворення відео, такі як рівень початкової якості, автоматичне завантаження, максимальна довжина та розмір буфера тощо.

Якщо підтримується відтворення HLS у поточному браузері Hls.isSupported(), то встановлюється джерело відео та виконується прикріплення медіа елементу до екземпляру - hlsInstance.loadSource(video.masterPlaylistUrl) та hlsInstance.attachMedia(videoRef.current). Після успішного парсингу маніфесту відео Hls.Events.MANIFEST_PARSED, стан компонента оновлюється з інформацією про доступні рівні якості відео. Крім того, екземпляр HLS зберігається в стані компонента для подальшого використання. У випадку, якщо підтримка HLS відсутня, але відео може бути відтворено за допомогою вбудованого плеєра браузера, встановлюється джерело відео прямо на медіа елемент - «videoRef.current.src = video.masterPlaylistUrl». Нарешті, під час видалення компонента, метод destroy() викликається для екземпляра HLS, щоб звільнити ресурси та зупинити всі активні процеси відтворення. Цей підхід дозволяє ефективно керувати відтворенням відео, враховуючи можливості браузера та налаштування відтворення відео через HLS.

Також важливим моментом є додання мануальної можливості перемикання якості зображення, яка реалізована за допомогою коду нижче:

```
const changeQuality = (selectedLevel: number) => {
  if (hls) hls.nextLevel = selectedLevel
  setVideoState(p => ({ ...p, selectedLevel }))
}
```

Перевіряємо чи існує обєкт hls, та задаємо нове значення якості, а також змінюємо стан, самого компонента, але і це не все необхідно додати слухач до самого обєкту hls, реалізуємо це за допомогою коду нижче:

```
hlsInstance.on(Hls.Events.LEVEL_SWITCHED, (_, { level }) => {
  setVideoState(p => ({ ...p, selectedLevel: level }))
})
```

Цей код відповідає за обробку події зміни рівня якості відео відтворюваного за допомогою HLS.js. Подія Hls.Events.LEVEL_SWITCHED спрацьовує при зміні рівня якості відео, наприклад, коли користувач переключається з одного рівня якості на інший (наприклад, з низької якості на високу або навпаки).

Не менш важливим є відслідковування помилок та їх вирішення під час відтворення відео, для цього також необхідно додати слухач подій, приклад коду наведено нижче:

```
hlsInstance.on(Hls.Events.ERROR, (_, data) => {
  if (data.fatal) {
   setVideoState(p => ({ ...p, isDisabled: true }))
   switch (data.type) {
    case Hls.ErrorTypes.NETWORK_ERROR:
     hlsInstance.startLoad()
     break
    case Hls.ErrorTypes.MEDIA_ERROR:
     hlsInstance.recoverMediaError()
     break
    default:
     hlsInstance.destroy()
     break
   }
  }
})
```

Даний код налаштовує обробку помилок для інстанції HLS.js, що забезпечує стабільну роботу відеоплеєра під час виникнення різних проблем. За допомогою методу «hlsInstance.on», ми підписуємось на події типу Hls.Events.ERROR, що сигналізують про виникнення помилок. Функція обробки помилок отримує два параметри: перший – це інформація про подію, яка не використовується безпосередньо у функції, а другий – це об'єкт даних, що містить детальну

інформацію про помилку. Якщо помилка є фатальною (тобто такою, що не дозволяє продовжити відтворення без втручання), стан відеоплеєра оновлюється, щоб позначити його як вимкнений (через функцію «setVideoState», яка змінює стан компонента).

Далі обробка помилки залежить від її типу. Якщо помилка стосується мережі Hls.ErrorTypes.NETWORK_ERROR, HLS.js спробує перезавантажити відео за допомогою методу hlsInstance.startLoad(). Якщо помилка пов'язана з медіа Hls.ErrorTypes.MEDIA_ERROR, HLS.js спробує відновити відтворення за допомогою методу hlsInstance.recoverMediaError(). У всіх інших випадках, коли тип помилки не відповідає вказаним, HLS.js інстанція буде знищена за допомогою методу hlsInstance.destroy(), що зупинить будь-які поточні процеси та звільнить ресурси. Цей підхід забезпечує гнучке і надійне оброблення помилок, підтримуючи відтворення відео в разі можливих проблем і забезпечуючи плавний користувацький досвід, загальний вигляд повністю завершеного плеєра зображено на рисунку 3.6.

Рисунок 3.6 – Повністю завершений плеєр

Продовжуючи створення функціональності відео, давайте розглянемо компонент VideoCreateModal, який призначений для створення нового відео через модальне вікно. Цей компонент використовує бібліотеки React, зокрема React Hook Form [51] для обробки форм та Zod [52] для валідації даних.

Компонент VideoCreateModal (рисунок 3.7) приймає два пропси: setShowModal та showModal, які використовуються для управління видимістю модального вікна. Всередині компонента використовується функція useTranslation з бібліотеки i18n для локалізації текстів, а також useRouter з Next.js для навігації після успішного створення відео.

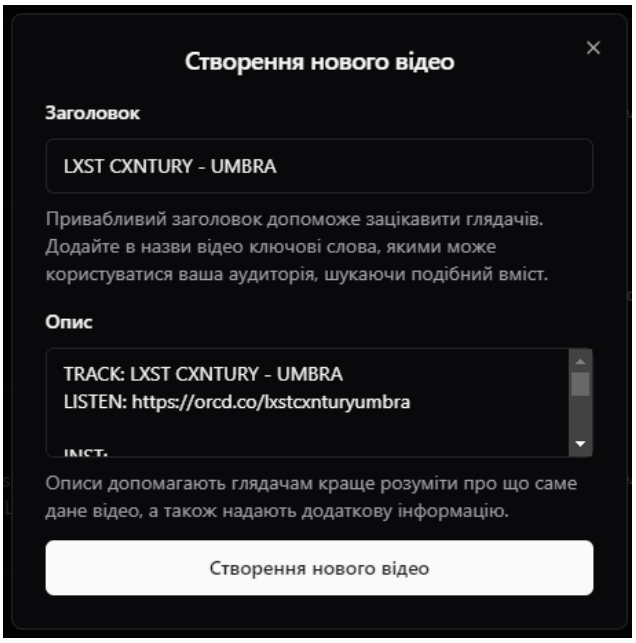


Рисунок 3.7 – Модальне вікно створення нового відео

Спочатку визначається схема валідації форми за допомогою бібліотеки Zod. Схема FormSchema містить два поля: title та description, кожне з яких має певні вимоги до мінімальної та максимальної довжини. Повідомлення про помилки локалізуються за допомогою функції t.

Далі створюється форма за допомогою useForm, де задаються початкові значення полів та резолвер для валідації схеми. Функція onSubmit визначає логіку обробки форми після її відправлення. Вона асинхронно викликає метод

createNewVideo сервісу VideoManagerService для створення нового відео на основі даних форми. Після успішного створення виконується перенаправлення на сторінку зі списком відео у студії за допомогою методу replace з useRouter, і модальне вікно закривається шляхом виклику setShowModal(false). У разі виникнення помилки вона обробляється та відображається за допомогою функції toastError. Приклад коду функції створення відео наведно нижче:

```
const onSubmit = async (videoCreateForm: z.infer<typeof FormSchema>) => {
  try {
   await VideoManagerService.createNewVideo(videoCreateForm)
   await replace('/studio/videos')
   setShowModal(false)
  } catch (e) {  toastError(e) }
}
```

Візуальна частина компонента складається з діалогового вікна, що відображається за умови, що showModal дорівнює true. Вміст модального вікна включає заголовок, форму для введення назви та опису відео, а також кнопку для підтвердження створення відео. Кожне поле форми обгорнуте компонентами FormField, які відповідають за рендеринг елементів форми та їх валідацію. Поля форми містять мітки FormLabel, поля введення Input або Textarea, описи FormDescription та повідомлення про помилки FormMessage.

Таким чином, компонент VideoCreateModal забезпечує зручний та інтуїтивно зрозумілий інтерфейс для створення нового відео з валідацією введених даних та обробкою помилок, що робить процес створення відео простим та надійним для користувача. Після успішного створення відео необхідно перейти в таблицю всіх відео користувача, про яку загадувалося раніше, та натиснути кнопку завантажити відео, далі виконується перехід до нового модального вікна завантаження відео.

Компонент VideoUploadModal (рисунок 3.8) відповідає за завантаження відеофайлів до створеного відео. Він приймає два пропси: відео, яке було створено, і функцію setVideo, щоб керувати станом відео. Використовується хук useTranslation для локалізації текстів, а також хуки useState для керування станами відеофайлу та процесу завантаження.

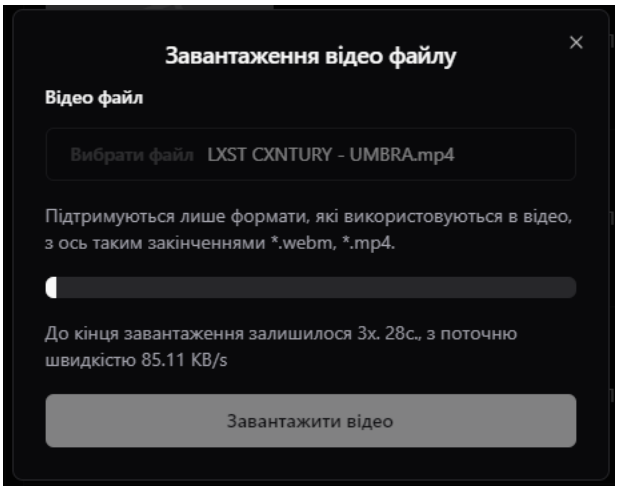


Рисунок 3.8 – Модальне вікно завантаження створеного відео

Основна логіка компонента зосереджена у функції onSubmit, яка виконується при натисканні кнопки завантаження. Спочатку генерується токен для завантаження відео за допомогою сервісу VideoManagerService. Потім створюється об'єкт FormData, до якого додається відеофайл. Починається завантаження відео через StorageService з використанням токену. В процесі завантаження оновлюється прогрес завантаження, відображаючи відсоток завантажених даних, швидкість завантаження та час, що залишився до завершення, код же даної функції зображено нижче:

```
const onSubmit = async () => {
  if (video)
   try {
    const {
     data: { token }
    } = await VideoManagerService.generateVideoUploadToken(video.id)

    const formData = new FormData()
    formData.append('file', videoFile!)
    const startTime = Date.now()

    await StorageService.uploadVideo(
     formData,
     token,
     undefined,
     ({ loaded, total }) => {
      const percentage = Math.round((loaded * 100) / (total || 0))
      const currentTime = Date.now()
      const timeElapsed = (currentTime - startTime) / 1000
      const uploadSpeed = loaded / timeElapsed
```

```
      const timeRemaining = ((total || 0) - loaded) / uploadSpeed
      setUploadProgress({ percentage, timeRemaining, uploadSpeed })
     }
    )
    setVideoFile(undefined)
    setUploadProgress(undefined)
    setVideo(undefined)
   } catch (e) {
    toastError(e)
   }
}
```

У візуальній частині компонента відображається модальне вікно з заголовком, полем для вибору відеофайлу та кнопкою для завантаження. Якщо процес завантаження триває, відображається індикатор прогресу та додаткова інформація про статус завантаження. Після завершення завантаження стан компонента очищується, відеофайл та прогрес завантаження скидаються, а модальне вікно закривається.

Цей компонент забезпечує зручний інтерфейс для завантаження відеофайлів, підтримуючи користувача на кожному етапі процесу та надаючи візуальний зворотний зв'язок про стан завантаження, що робить процес завантаження відео простим та ефективним.

3.5 Інтеграція з бекендом

Інтеграція з бекендом у сучасних програмно-технічних системах є важливим аспектом, який забезпечує взаємодію між клієнською і серверною частиною. Вона дозволяє реалізувати різні функціональні можливості, такі як авторизація, управління користувачами, робота з відео, отрмання популярних відео тощо. Для успішної інтеграції з бекендом необхідно встановити Axios [53] та додати до нього інтерцептори, які допоможуть управляти автентифікацією та оновленням токенів доступу. Це забезпечує безперервну роботу клієнтської частини навіть у випадку закінчення терміну дії токена.

Функція setupAxiosInterceptors [54] використовує store для отримання поточного стану автентифікації та налаштовує інтерцептори запитів та відповідей,

для цього її потрібно викликати в компоненті app. Якщо отримано відповідь з кодом 401, інтерсептор намагається оновити токен доступу за допомогою AuthService. Якщо оновлення не вдається, користувач буде автоматично розлогінений. Також для приклада розглянемо, як здійснюється інтеграція з бекендом за допомогою сервісів, одним з яких є AuthService. Код якого можна глянути нижче:

```
export const AuthService = {
  async login(loginBody: ILoginRequest) {
   return $axios.post<ILoginResponse>('auth/login', loginBody, { baseURL })
  },

  async signup(signupData: ISignUpRequest) {
   return $axios.post<ILoginResponse>('auth/signup', signupData, { baseURL })
  },

  async recoveryPass(data: IRecoveryPassRequest) {
   return $axios.post<IErrorResponse>('auth/recovery/create-token', data, {
    baseURL
   })
  },

  async passReset(data: any) {
   return $axios.post<IErrorResponse>('auth/recovery/reset-password', data, {
    baseURL
   })
  },

  async refreshAccessToken() {
   return axios.get<IRefreshAccessTokenResponse>('auth/refresh', { baseURL })
  },

  async logout(accessToken?: string) {
   return axios.get('auth/logout', {
    baseURL,
    ...(accessToken
     ? { headers: { Authorization: `Bearer ${accessToken}` } }
     : undefined)
   })
  }
}
```

Сервіс AuthService відповідає за аутентифікацію та управління доступом користувачів. Він реалізує набір методів, які взаємодіють з сервером через HTTP-запити. Для цього використовується бібліотека axios, яка забезпечує зручний інтерфейс для виконання HTTP-запитів. Базова URL-адреса сервера зберігається у

змінній baseURL, що дозволяє легко змінювати адресу сервера автооризації, не змінюючи код кожного методу. Розглянемо основні методи

Метод login приймає об'єкт loginBody з даними для входу (логін і пароль), відправляє POST-запит до кінцевої точки auth/login і повертає відповідь сервера про успішний вхід або помилку. Метод signup працює аналогічно, але для реєстрації нових користувачів, відправляючи об'єкт signupData до auth/signup. Метод recoveryPass створює токен для відновлення пароля, приймаючи об'єкт data і відправляючи його до auth/recovery/create-token. Метод passReset дозволяє скинути пароль, приймаючи новий пароль і токен відновлення, і відправляє їх до auth/recovery/reset-password. Метод refreshAccessToken оновлює токен доступу через GET-запит до auth/refresh. Метод logout дозволяє вийти з облікового запису, відправляючи GET-запит до auth/logout, з опціональним параметром accessToken у заголовку запиту.

Кожен з методів AuthService забезпечує взаємодію з відповідними кінцевими точками бекенду, забезпечуючи аутентифікацію, реєстрацію, відновлення паролів та інші операції, пов'язані з управлінням доступом користувачів. Цей підхід дозволяє чітко розділити логіку взаємодії з серверною та клієнтською частиною, роблячи код більш організованим і легким для підтримки.

3.6 Розгортання клієнтської частини на сервері

Розгортання клієнтської частини на сервері є важливим кроком у процесі розробки веб-додатку, який забезпечує доступність додатку для кінцевих користувачів. Цей процес включає кілька етапів: збірка додатку, налаштування сервера, встановлення необхідного програмного забезпечення та розгортання додатку.

Після успішної збірки серверу, необхідно підготувати сервер для запуски системи. Для цього можна використовувати операційну систему Ubuntu, яка є однією з найпопулярніших і найнадійніших платформ для розгортання веб-

додатків. Встановлення Ubuntu на сервер може бути виконане за допомогою різних методів, але в свою чергу реалізовано за допомоги завантаження ISO-образу та його встановлення на переносний носій.

Після встановлення Ubuntu на сервер, необхідно встановити Docker. Docker є контейнерною платформою, яка дозволяє автоматизувати розгортання додатків у незалежних від системного середовища контейнерах. Це забезпечує ізоляцію додатку та спрощує процес його розгортання і масштабування. Для встановлення Docker на Ubuntu можна виконати наступні команди:

```
sudo apt update
sudo apt install docker.io -y
sudo systemctl start docker
sudo systemctl enable docker
```

Наступним кроком є створення Dockerfile для додатку. Dockerfile визначає інструкції для створення Docker-образу, який містить всі необхідні залежності та конфігурації для запуска додатку. Нижче наведено приклад коду Dockerfile для розгортання системи:

```
FROM node:20.11.0 as build
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM node:20.11.0-slim
RUN apt update && apt install libssl-dev -y --no-install-recommends
WORKDIR /app
COPY --chown=node:node --from=build /app/.next ./next
COPY --chown=node:node --from=build /app/.env .env
COPY --chown=node:node --from=build /app/package*.json ./
RUN npm install --omit=dev
ENV NODE_ENV production
EXPOSE 3000
CMD ["npm", "start"]
```

Перший етап в Dockerfile, який виконується на базовому образі node, включає встановлення всіх необхідних залежностей та побудову додатку. Цей

етап використовує робочий каталог /app, копіює файли package.json та package-lock.json, встановлює залежності за допомогою команди npm install, а потім копіює всі інші файли додатку і виконує команду npm run build для збірки Next.js додатку.

Другий етап використовує більш легкий образ node-slim для мінімізації розміру фінального образу. Тут виконується оновлення списку пакетів та встановлення бібліотеки libssl-dev. Потім копіюються зібрані файли з першого етапу, встановлюються необхідні для продакшн середовища залежності за допомогою команди npm install --omit=dev, встановлюється змінна середовища NODE_ENV у значення production, відкривається порт 3000 для доступа до додатку, і запускається додаток за допомогою команди npm start.

Для фінального створення Docker-образу та запуску контейнера необхідно виконати наступні команди в терміналі:

docker build -t uatube_client

docker run -d -p 3000:3000 uatube_client

Ці команди створять Docker-образ з тегом my-next-app та запустять його у новому контейнері, відкривши порт 3000 для доступа до додатку. Після цього клієнтська частина буде доступна для користувачів через веб-браузер.

Таким чином, процес розгортання клієнтської частини на сервері включає підготовку сервера, встановлення Docker, створення Dockerfile, збірку Docker-образу та запуск контейнера. Використання Docker спрощує цей процес, забезпечуючи ізоляцію додатку та його залежностей, що підвищує надійність і масштабованість вашого веб-додатку.

3.7 Висновки

У межах розділу 3 було проведено детальний опис усіх етапів створення клієнтської частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту.

Початковим кроком було створення та налаштування проєкту, де ми визначили основні інструменти та технології, необхідні для реалізації завдань. Далі, ми розробили архітектуру системи, що включала детальний огляд компонентів та їх взаємодій, забезпечуючи основу для стабільної та масштабованої системи.

Наступним кроком стала розробка інтерфейсу, яка передбачала створення зручного та інтуїтивно зрозумілого користувацького інтерфейсу, що сприяє кращому взаємозв’язку між користувачами та системою. Потім було реалізовано розробку функціоналу, який включав усі необхідні можливості для роботи з відеоконтентом, забезпечуючи повний цикл обробки, зберігання та передачі відеоданих.

Інтеграція з бекендом була важливою складовою, що дозволила забезпечити безперебійну взаємодію між клієнтською та серверною частинами, використовуючи приклад AuthService для демонстрації основних підходів. Розгортання клієнтської частини на сервері було завершальним етапом, що включало встановлення необхідного програмного забезпечення, створення Docker-образів та запуск додатку на сервері.

ВИСНОВКИ

У роботі за результатами виконаних теоретичних та практичних досліджень було створено та розроблено програмно-технічну систему обробки, зберігання та передачі відеоконтенту, з особливим акцентом на клієнтську частину.

У першому розділі проведено аналіз існуючих рішень на ринку, що дозволило визначити основні переваги та недоліки наявних програмно-технічних систем для роботи з відеоконтентом. Було здійснено порівняльний аналіз різних програмних підходів, які використовуються для зберігання та передачі відеоданих, що допомогло сформувати вимоги для системи та вибрані найбільш оптимальні технології для її реалізації.

У другому розділі проведено детальне дослідження архітектури системи, вибору програмних та апаратних засобів розробки клієнтської частини. Було розглянуто архітектуру клієнтської частини, визначено основні компоненти та їх взаємодії. Крім того, обґрунтовано вибір мов програмування, фреймворків та бібліотек, а також вибрано апаратні засоби, які оптимально підходять для виконання вимог програми.

У третьому розділі описано процес розробки клієнтської частини системи. Було детально розглянуто створення та налаштування проєкту, розробку архітектури та інтерфейсу, реалізацію функціоналу та інтеграцію з бекендом. Окрему увагу приділено розгортанню клієнтської частини на сервері та оптимізації додатку. Завдяки використанню сучасних інструментів та технологій, таких як Docker та Next.js, вдалося досягти високої продуктивності та ефективності роботи системи, забезпечуючи при цьому зручність та надійність для кінцевих користувачів.

Загалом, результати проведених досліджень і розробки показали, що запропонована програмно-технічна система є ефективною та готовою до

практичного застосування, відповідаючи сучасним вимогам до обробки, зберігання та передачі відеоконтенту.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Що таке адаптивність веб сайту під різні пристрої – WebTune. URL: https://webtune.com.ua/statti/internet-marketing/yak-pereviryty-adaptyvnist-za-dopomogoyu-brauzera (дата звернення 29.12.2023)

2. Ключові принципи для створення ефективної навігації на веб-сайті. URL: https://markweb.pro/kljuchovi-principi-dlya-stvorennya-efektivnoi-navigacii-na-veb-sajti-poradi-z-ui-ux-dizajnu (дата звернення 30.12.2023)

3. UX дослідження: методи аналізу та визначення потреб користувачів. URL: https://blog.ithillel.ua/articles/best-practices-for-determining-user-needs (дата звернення 03.01.2024)

4. What is Design Thinking? — updated 2024 | IxDF. URL: https://www.interaction-design.org/literature/topics/design-thinking (дата звернення 27.12.2023)

5. Lean UX | Everything you need to know about this methodology. URL: https://www.plainconcepts.com/lean-ux-methodology (дата звернення 01.02.2024)

6. What is User Centered Design (UCD)? — updated 2024 | IxDF. URL: https://www.interaction-design.org/literature/topics/user-centered-design (дата звернення 22.01.2024)

7. What is user analytics? A guide to user analysis | Fullstoryю. URL: https://www.fullstory.com/user-analytics (дата звернення 30.12.2023)

8. What is UX testing with example | BrowserStack. URL: https://www.browserstack.com/guide/what-is-ux-testing (дата звернення 15.01.2024)

9. When to use Sentry: error management and real-time monitoring. URL: https://www.mytaskpanel.com/when-to-use-sentry (дата звернення 12.01.2024)

10. What is a Rollbar? | TriCity New Balance. URL: https://tricitynewbalance.wordpress.com/new-balance-information/what-is-a-rollbar (дата звернення 14.01.2024)

11. The Best Operating System for Programming. URL: https://www.naukri.com/code360/library/the-best-operating-system-for-programming (дата звернення 28.03.2024)

12. What is a Linux Server and Why use it – GeeksforGeeks. URL: https://www.geeksforgeeks.org/what-is-a-linux-server-and-why-use-it (дата звернення 09.01.2024)

13. Мови програмування та сфери їх застосування | JUNGO. URL: https://blog.jungo.dev/uk/2020/12/yazyki-programmirovaniya-i-sfery-ih-primeneniya-2 (дата звернення 06.01.2024)

14. TypeScript це JavaScript на стероїдах | Друкарня URL: https://drukarnia.com.ua/articles/typescript-ce-javascript-na-steroyidakh-uAxV2 (дата звернення 08.01.2024)

15. A Comprehensive Guide to Next.js. Introduction to Next.js | by Ahmed Abu Bakr | Medium URL: https://medium.com/@ahmed.nums345/a-comprehensive-guide-to-next-js-5f3b03b49def (дата звернення 06.01.2024)

16. Basics of ReactJS. Basic knowledge about ReactJS | by Ashen Kavinda | Medium URL: https://medium.com/@ashenkavinda/basics-of-reactjs-79107d19adfb (дата звернення 08.01.2024)

17. Epyc — Вікіпедія. URL: https://uk.wikipedia.org/wiki/Epyc (дата звернення 09.01.2024)

18. Xeon — Вікіпедія. URL: https://uk.wikipedia.org/wiki/Xeon (дата звернення 09.01.2024)

19. Процесор Intel XEON 4 Core E3-1225 V5 3.30GHz (SR2LJ). URL: https://hard.rozetka.com.ua/ua/325390303/p325390303 (дата звернення 10.01.2024)

20. Графічний процесор — Вікіпедія. URL: https://uk.wikipedia.org/wiki/Графічний_процесор (дата звернення 11.01.2024)

21. NVIDIA QUADRO K420/K600 1GB (713379-001, 0V5WK5, VCQK600-T) High Profile URL: https://servak.com.ua/ua/komplektujuschie-k-

serveram/videokarta/videokarta-nvidia-quadro-k420-k600-1gb-high-profile-pn-713379-001.html (дата звернення 11.01.2024)

22. ECC-пам'ять — Вікіпедія. URL: https://uk.wikipedia.org/wiki/ECC-пам%27ять (дата звернення 12.01.2024)

23. Пам'ять для сервера Hynix DDR4-2133 128Gb (8x16Gb) ECC Registered Memory Kit | HARD.kiev.ua. URL: https://hard.kiev.ua/pam-yat-dlya-servera-hynix-ddr4-2133-128gb-8x16gb-ecc-registered-memory-kit (дата звернення 12.01.2023)

24. SSD – Вікіпедія. URL: https://uk.wikipedia.org/wiki/SSD (дата звернення 13.01.2024)

25. Understanding SSD Technology: NVMe, SATA, M.2- Kingston Technology. URL: https://www.kingston.com/en/ssd/what-is-nvme-ssd-technology (дата звернення 13.01.2024)

26. Жорсткий Диск EO0400JEFPE HP Enterprise 400GB 2.5" SAS URL: https://bestock.com.ua/zhorstkiy-disk-eo0400jefpe-hp-g8-g9-400gb-2.5-sas-12g-wi-ssd (дата звернення 13.01.2024)

27. P10S-M WS | Материнські плати | ASUS Україна. URL: https://www.asus.com/ua-ua/motherboards-components/motherboards/workstation/p10s-m-ws/ (дата звернення 13.01.2023)

28. Understanding SSR in Next.js and Its Benefits. URL: https://medium.com/@markminj/understanding-ssr-in-next-js-and-its-benefits-e54ffed48294 (дата звернення 14.03.2024)

29. Що таке SEO-просування сайту: основи SEO. URL: https://ideadigital.agency/blog/shcho-take-seo-i-navishcho-potribna-poshukova-optimizaciya/ (дата звернення 16.03.2024)

30. App Router | Next.js. URL: https://nextjs.org/docs/app (дата звернення 20.03.2024)

31. Pages Router | Next.js. URL: https://nextjs.org/docs/pages (дата звернення 20.03.2024)

32. Redux Persist Store. URL: https://blog.logrocket.com/persist-state-redux-persist-redux-toolkit-react (дата звернення 21.03.2024)

33. Redux - A JS library for predictable and maintainable global state management | Redux URL: https://redux.js.org/ (дата звернення 20.03.2024)

34. The Role of Website Themes in Design and Engagement URL: https://debutify.com/blog/impacts-of-website-themes-on-user-experience/ (дата звернення 19.03.2024)

35. What Is Cross-Site Request Forgery (CSRF) and How Does It Work? | Synopsys. URL: https://www.synopsys.com/glossary/what-is-csrf.html (дата звернення 12.04.2024)

36. Cross-site request forgery. URL: https://en.wikipedia.org/wiki/Cross-site_request_forgery (дата звернення 24.04.2024)

37. JSON Web Token – Вікіпедія. URL: https://uk.wikipedia.org/wiki/JSON_Web_Token (дата звернення 25.04.2024)

38. HTTPS – Вікіпедія. URL: https://uk.wikipedia.org/wiki/HTTPS (дата звернення 25.04.2024)

39. Атака «людина посередині». URL: https://uk.wikipedia.org/wiki/Атака_«людина_посередині» (дата звернення 29.04.2024)

40. HTTP Live Streaming – Wikipedia. URL: https://en.wikipedia.org/wiki/HTTP_Live_Streaming (дата звернення 01.04.2024)

41. HTTP Live Streaming (HLS) - Apple Developer. URL: https://developer.apple.com/streaming/ (дата звернення 02.04.2024)

42. Advanced Encryption Standard — Вікіпедія. URL: https://uk.wikipedia.org/wiki/Advanced_Encryption_Standard (дата звернення 03.04.2024)

43. What Is NPM? What are Popular Node Packages? | Built In. URL: https://builtin.com/software-engineering-perspectives/npm (дата звернення 05.04.2024)

44. Tailwind CSS: What It Is, Why Use It & Examples. URL: https://blog.hubspot.com/website/what-is-tailwind-css (дата звернення 10.04.2024)

45. Shadcn UI adoption guide: Overview, examples, and alternatives – LogRocket Blog. URL: https://blog.logrocket.com/shadcn-ui-adoption-guide/ (дата звернення 12.04.2024)

46. Інтернаціоналізація та локалізація — Вікіпедія. URL: https://en.wikipedia.org/wiki/Internationalization_and_localization (дата звернення 12.04.2024)

47. CI/CD — Вікіпедія. URL: https://uk.wikipedia.org/wiki/CI/CD (дата звернення 12.04.2024)

48. Components: <Link> | Next.js URL: https://nextjs.org/docs/pages/api-reference/components/link (дата звернення 13.02.2024)

49. Implementing Dark and Light Mode Themes in Next.js: Medium. URL: https://medium.com/@aashekmahmud/implementing-d (дата звернення 13.04.2024)

50. HLS.js is a JavaScript library that plays HLS in browsers with support for MSE. URL: https://github.com/video-dev/hls.js/ (дата звернення 13.02.2024)

51. React Hook Form - Simple React forms validation. URL: https://www.react-hook-form.com/get-started (дата звернення 11.04.2024)

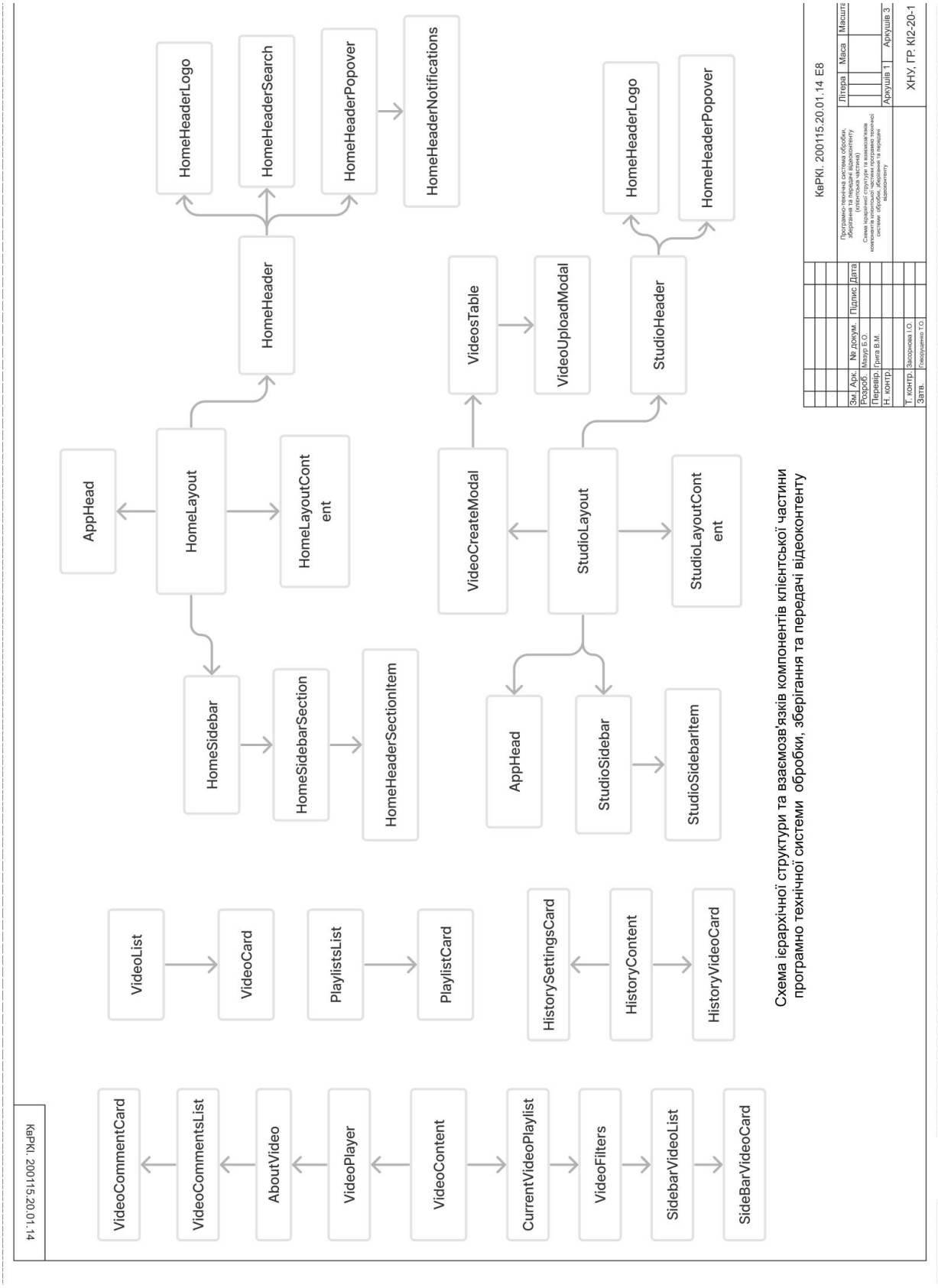
52. Zod | Documentation. URL: https://zod.dev/ (дата звернення 13.02.2024)

53. Getting Started | Axios Docs. URL: https://axios-http.com/docs/intro (дата звернення 13.01.2024)

54. client/src/api/interceptor.ts URL:https://github.com/ua-tube/client/blob/master/src/api/interceptor.ts (дата звернення 13.05.2024)

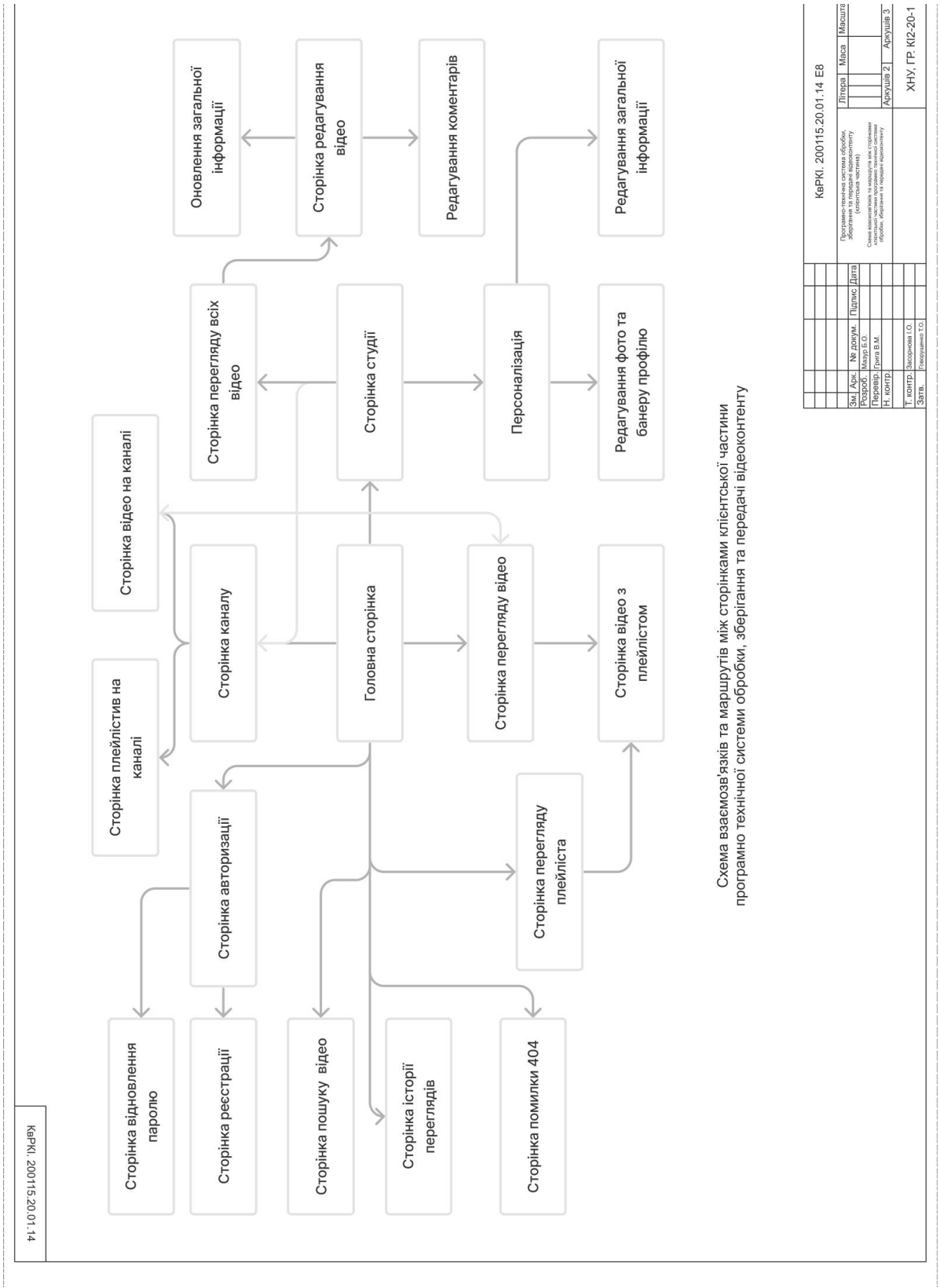
Додаток А

Копія креслення «Схема ієрархічної структури та взаємозв'язків компонентів клієнтської частини програмно технічної системи обробки, зберігання та передачі відеоконтенту»



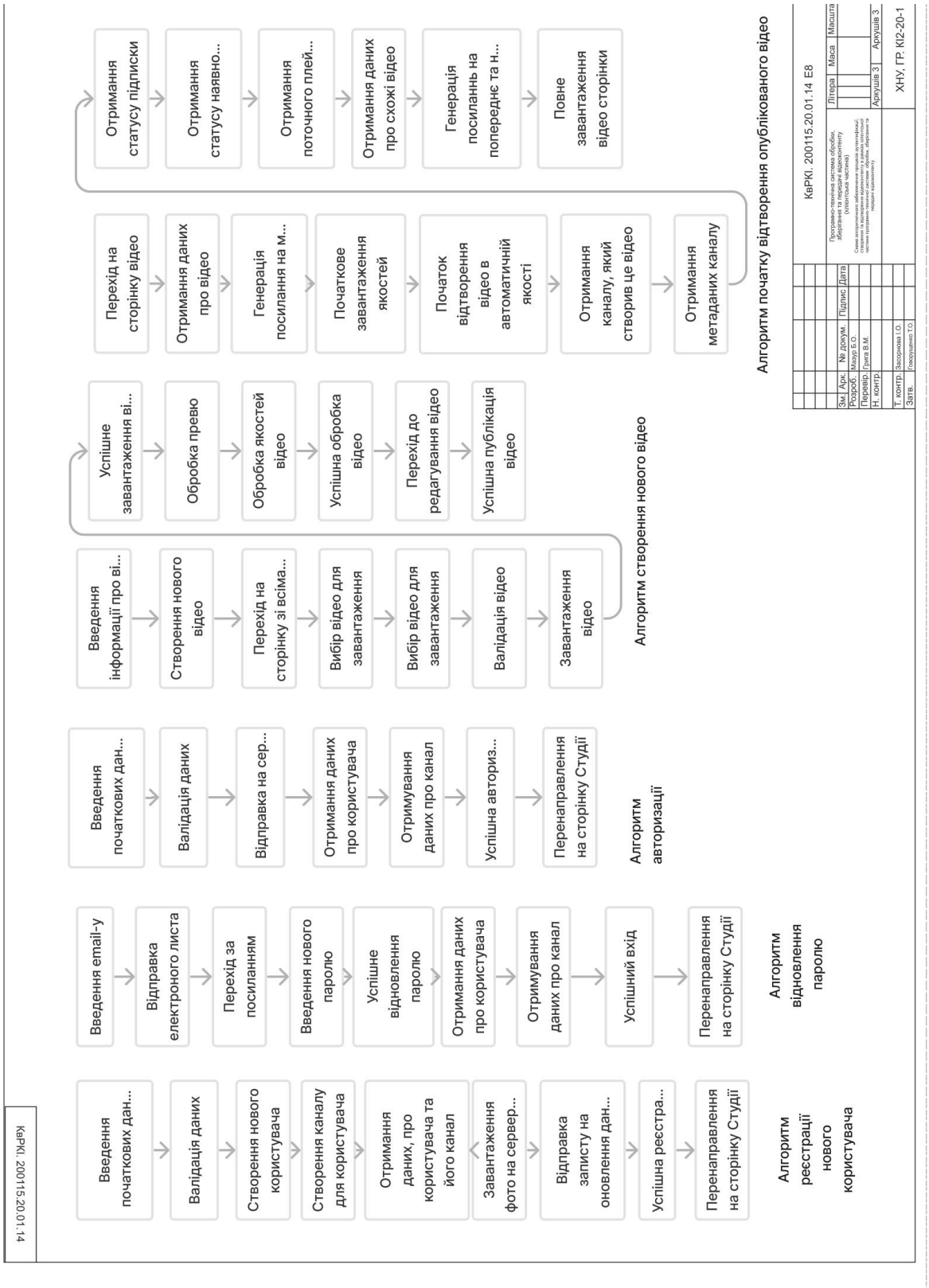
Додаток Б

Копія креслення «Схема взаємозв'язків та маршрутів між сторінками клієнтської частини програмно технічної системи обробки, зберігання та передачі відеоконтенту»



Додаток В

Копія креслення «Схема алгоритмічного забезпечення процесів аутентифікації, створення та відтворення відеоконтенту в рамках клієнтської частини програмно-технічної системи обробки, зберігання та передачі відеоконтенту»



Додаток Г

Лістинг коду компонента відеоплеєра

VideoPlayer.tsx

```
import { cn, formatDuration,
getVideoUrl, writeVideoUrl } from
'@/utils'
import { FC, useCallback, useEffect,
useRef, useState } from 'react'
import * as SliderPrimitive from
'@radix-ui/react-slider'
import { IVideo, UseState } from
'@/interfaces'
import { useSidebarContext } from
'@/providers'
import { useTranslation } from 'next-
i18next'
import { useRouter } from
'next/router'
import Link from 'next/link'
import Hls from 'hls.js'
import {
  Button,
  ContextMenu,
  ContextMenuContent,
  ContextMenuItem,
  ContextMenuTrigger,
  DynamicIcon,
  HoverCard,
  HoverCardArrow,
  HoverCardContent,
  HoverCardTrigger,
  Slider,
  Switch,
  Tooltip,
  TooltipArrow,
  TooltipContent,
  TooltipProvider,
  TooltipTrigger,
  Progress
} from '@/components'

interface IVideoPlayerProps {
  video: Pick<
   IVideo,
   'nextId' | 'prevId' |
'masterPlaylistUrl' | 'id' | 'status'
  >
  autoPlay?: boolean
  cinemaMode: boolean
  setCinemaMode: UseState<boolean>
}

interface IHlsLevel {
  height: number
  width: number
}

interface IVideoState {
  selectedLevel: number
  levels?: Array<IHlsLevel>
  speed: number
  volume: number
  duration: number
  bufferedCount: number
  isLoading: boolean
  currentTime: number
  autoPlayNext: boolean
  isFullScreen: boolean
  showNavigationMenu: boolean
  showLoadingAnimation: boolean
  isLooped: boolean
  isDisabled: boolean
}

const VideoPlayer: FC<
  IVideoPlayerProps
> = ({
     video,
     autoPlay,
     setCinemaMode,
     cinemaMode
    }) => {

  const { t } =
useTranslation('videos')
  const { push, query } = useRouter()
  const { isOpen } =
useSidebarContext()

  const [hls, setHls] = useState<Hls |
null>(null)
  const videoRef =
useRef<HTMLVideoElement>(null)
  const divRef =
useRef<HTMLDivElement>(null)

  const [menuOpen, setMenuOpen] =
useState<boolean>(false)
  const [videoState, setVideoState] =
useState<IVideoState>({
   autoPlayNext: !!video.nextId,
   selectedLevel: -1,
   levels: [],
   isLooped: false,
   duration: 0,
   isLoading: true,
   speed: 1,
```

```
    volume: 0.5,
    isFullScreen: false,
    currentTime: 0,
    bufferedCount: 0,
    showLoadingAnimation: false,
    showNavigationMenu: true,
    isDisabled: false
   })

  const onProgressHandler =
useCallback(() => {
    if (videoRef.current) {
     const { buffered, currentTime } =
videoRef.current
     let bufferedCount = 0

     for (let i = 0; i <
buffered.length; i++) {
      if (buffered.end(i) >
currentTime) {
       if (buffered.start(i) <
currentTime)
        bufferedCount +=
buffered.end(i) - currentTime
       else bufferedCount +=
buffered.end(i) - buffered.start(i)
      }
     }

     setVideoState(prevState => ({
      ...prevState,
      bufferedCount
      // isLoading: currentTime >
bufferedCount
     }))
    }
  }, [setVideoState])

  const onTimeUpdateHandler =
useCallback(
    (time?: number) => {
     if (videoRef.current) {
      if (time)
videoRef.current.currentTime = time
      setVideoState(p => ({
       ...p,
       currentTime: time ||
videoRef.current!.currentTime
      }))
     }
    },
    [setVideoState]
  )

  const changeQuality =
(selectedLevel: number) => {
    if (hls) hls.nextLevel =
selectedLevel
    setVideoState(p => ({ ...p,
selectedLevel }))
  }

  const showPlayAnimation =
useCallback(() => {
    setVideoState(s => ({ ...s,
showLoadingAnimation: true }))
    setTimeout(
     () => setVideoState(s => ({ ...s,
showLoadingAnimation: false })),
     1000
    )
  }, [setVideoState])

  const togglePlay = useCallback(
   (disableAnimation?: boolean) => {
    if (videoRef.current) {
     if (videoRef.current.paused)
videoRef.current.play().catch()
     else videoRef.current.pause()
     if (!disableAnimation)
showPlayAnimation()
    }
   },
   [showPlayAnimation]
  )

  const changeSpeed = useCallback(
   (speed: number) => {
    if (videoRef.current) {
     videoRef.current.playbackRate =
speed
     setVideoState(p => ({ ...p, speed
}))
    }
   },
   [setVideoState]
  )

  const changeVolume = useCallback(
   (volume: number) => {
    if (videoRef.current && volume <
1.01 && volume > -0.01) {
     videoRef.current.volume = volume
     setVideoState(p => ({ ...p,
volume }))
     sessionStorage.setItem(
      'video-state',
      JSON.stringify({ ...videoState,
volume })
     )
    }
   },
   [setVideoState, videoState]
  )

  const toggleMute = useCallback(
   () =>
changeVolume(videoState.volume !== 0 ?
0 : 0.4),
   [videoState.volume, changeVolume]
  )

  const toggleFullScreen =
useCallback(() => {
   if (divRef.current) {
    if (!videoState.isFullScreen) {
```

```
      divRef.current
       ?.requestFullscreen()
       ?.catch(reason =>
console.log(reason))
      setVideoState(p => ({ ...p,
isFullScreen: true }))
    } else {

document?.exitFullscreen()?.catch(reas
on => console.log(reason))
      setVideoState(p => ({ ...p,
isFullScreen: false }))
    }
   }
  }, [videoState.isFullScreen,
setVideoState])
  const onVideoEnd = useCallback(async
() => {
   if (videoState.autoPlayNext &&
videoState.isLooped &&
videoRef.current) {
    videoRef.current.loop = false
    setVideoState(p => ({ ...p,
isLooped: false }))
   }
   if (videoState.autoPlayNext &&
video?.nextId)
    await push(
     getVideoUrl(
      video?.nextId,
      undefined,
      query?.listId ? (query.listId as
string) : undefined,
       true
     )
    )
  }, [
   videoState.autoPlayNext,
   push,
   query.listId,
   videoState.isLooped,
   setVideoState,
   video?.nextId
  ])

  const toggleRepeat = useCallback(()
=> {
   if (videoRef.current) {
    videoRef.current.loop =
!videoRef.current.loop
    setVideoState(p => ({
     ...p,
     isLooped: videoRef.current!.loop,
     autoPlayNext: false
    }))
   }
  }, [setVideoState])
  const toggleAutoPlayNext =
useCallback(() => {
   videoState.isLooped &&
toggleRepeat()
   sessionStorage.setItem(
    'video-state',
    JSON.stringify({ ...videoState,
autoPlayNext: !videoState.autoPlayNext
})
   )
   setVideoState(p => ({ ...p,
autoPlayNext: !p.autoPlayNext }))
  }, [toggleRepeat, setVideoState,
videoState])

  const buttonsCallBack = useCallback(
   (event: KeyboardEvent) => {
    if (videoState.isFullScreen) {
     const key =
event.key.toLowerCase()
     switch (key) {
      case 'f':
       toggleFullScreen()
       break
      case ' ':
       togglePlay()
       break
      case 'm':
       toggleMute()
       break
      case 'arrowup':
       changeVolume(videoState.volume
+ 0.05)
       break
      case 'arrowdown':
       changeVolume(videoState.volume
- 0.05)
       break
      case 'arrowleft':
onTimeUpdateHandler(videoState.current
Time - 5)
       break
      case 'arrowright':

onTimeUpdateHandler(videoState.current
Time + 5)
       break
     }
    }
   },
   [
    videoState.volume,
    videoState.currentTime,
    toggleFullScreen,
    togglePlay,
    toggleMute,
    changeVolume,
    onTimeUpdateHandler,
    videoState.isFullScreen
   ]
  )
  const fullScreenChangeListener =
useCallback(
   (event: Event) => {
    event.preventDefault()
    videoState.isFullScreen &&
    setVideoState(p => ({
     ...p,
     isFullScreen:
```

```
      (document.fullscreenEnabled ?
document.fullscreenElement : null) !==
      null
    }))
   },
   [videoState.isFullScreen,
setVideoState]
  )
  const onLoadingListener =
useCallback(
   (isLoading: boolean) =>
setVideoState(p => ({ ...p, isLoading
})),
   [setVideoState]
  )

  useEffect(() => {

document.addEventListener('keydown',
buttonsCallBack)

document.addEventListener('fullscreenc
hange', fullScreenChangeListener)

   if (videoRef.current) {

videoRef.current.addEventListener('can
play', () =>
     onLoadingListener(false)
    )
   }
   return () => {

document.removeEventListener('keydown'
, buttonsCallBack)

document.removeEventListener('fullscre
enchange', fullScreenChangeListener)

    if (videoRef.current) {

videoRef.current.removeEventListener('
canplay', () =>
      onLoadingListener(false)
     )
    }
   }
  }, [videoState])

  useEffect(() => {
   let timeout: any

   const displayOpa = () => {
    setVideoState(p => ({ ...p,
showNavigationMenu: true }))

    clearTimeout(timeout)

    timeout = setTimeout(() => {
     setVideoState(p => ({ ...p,
showNavigationMenu: false }))
    }, 5000)
   }

divRef.current?.addEventListener('mous
emove', displayOpa)
   return () => {
divRef.current?.removeEventListener('m
ousemove', displayOpa)
   }
  }, [])

  useEffect(() => {
   const hlsInstance = new Hls({
    startLevel: -1,
    autoStartLoad: true,
    maxBufferLength: 60,
    maxBufferSize: 60 * 1000 * 1000,
    abrEwmaDefaultEstimate: 5 * 1000 *
1000
   })
   if (Hls.isSupported() &&
videoRef?.current) {

hlsInstance.loadSource(video?.masterPl
aylistUrl!)

hlsInstance.attachMedia(videoRef.curre
nt!)


hlsInstance.on(Hls.Events.MANIFEST_PAR
SED, () =>
     setVideoState(p => ({
      ...p,
      levels: hlsInstance.levels,
      selectedLevel:
hlsInstance.currentLevel
     }))
    )

    hlsInstance.on(Hls.Events.ERROR,
(_, data) => {
     if (data.fatal) {
      setVideoState(p => ({ ...p,
isDisabled: true }))
      switch (data.type) {
       case
Hls.ErrorTypes.NETWORK_ERROR:
        console.error('Fatal network
error:', data)
        hlsInstance.startLoad()
        break
       case
Hls.ErrorTypes.MEDIA_ERROR:
        console.error('Fatal error of
the media:', data)

hlsInstance.recoverMediaError()
        break
       default:
        console.error('Unknown fatal
error:', data)
        hlsInstance.destroy()
        break
      }
```

```
      }
    })


hlsInstance.on(Hls.Events.LEVEL_SWITCH
ED, (_, { level }) => {
     setVideoState(p => ({ ...p,
selectedLevel: level }))
    })

    setHls(hlsInstance)
   } else if
(videoRef.current?.canPlayType('applic
ation/vnd.apple.mpegurl')) {
    videoRef.current.src =
video?.masterPlaylistUrl!
   }

   return () => {
    if (hlsInstance)
hlsInstance.destroy()
   }
  }, [video])

  useEffect(() => {
   setVideoState(p => ({
    ...p,
    bufferedCount: 0,
    selectedQuality: -1,
    isDisabled: false
   }))

   const previousVideoStateStr =
sessionStorage.getItem('video-state')
   const previousVideoState =
previousVideoStateStr
    ?
(JSON.parse(previousVideoStateStr) as
IVideoState)
    : null

   !videoState.autoPlayNext &&
   previousVideoState?.autoPlayNext &&
   toggleAutoPlayNext()
   typeof previousVideoState?.volume
!== 'undefined' &&

changeVolume(previousVideoState.volume
|| 0.5)
   previousVideoState?.speed &&
changeSpeed(previousVideoState?.speed
|| 1)
  }, [video])

  return (
   <div
    ref={divRef}
    onDoubleClick={toggleFullScreen}
    className={cn(
     'group/main relative',
     !cinemaMode ? 'rounded-lg' : 'h-
[80vh] mb-4 bg-black/80',
     cinemaMode &&
!videoState.isFullScreen && ' border-
b-2'
    )}
   >
    <ContextMenu>
     <ContextMenuTrigger
className="size-full flex items-center
justify-center overflow-y-hidden">
      <video
       ref={videoRef}
       controls={false}
       className={cn(
        'bg-black/80 ',
        !cinemaMode
         ? ' w-full aspect-video
rounded-lg'
         : 'h-full object-center'
       )}
       onClick={() => togglePlay()}
       onLoadStart={() =>
onLoadingListener(true)}
       onLoadedData={() =>
onLoadingListener(false)}
       onProgress={onProgressHandler}
       onTimeUpdate={() =>
onTimeUpdateHandler()}
       onLoadedMetadata={() => {
        setVideoState(p => ({
         ...p,
         duration:
videoRef.current?.duration || 0
        }))
        autoPlay && togglePlay(true)
       }}
       onEnded={onVideoEnd}
      />
     </ContextMenuTrigger>

     <ContextMenuContent>
      {video?.nextId && (
       <ContextMenuItem
        className="flex items-center
justify-between"
        onClick={toggleAutoPlayNext}
       >
        <div className="items-center
flex space-x-2">
         <DynamicIcon name="arrow-
right-circle" />
         <span
children={t('autoPlay')} />
        </div>
        {videoState.autoPlayNext &&
<div className="checkedIcon" />}
       </ContextMenuItem>
      )}

      <ContextMenuItem
       className="flex items-center
justify-between"
       onClick={toggleRepeat}
      >
       <div className="items-center
```

```
flex space-x-2">
        <DynamicIcon name="repeat" />
        <span children={t('repeat')}
/>
       </div>
       {videoState.isLooped && <div
className="checkedIcon" />}
      </ContextMenuItem>

      <ContextMenuItem
       className="flex items-center
justify-between"
       onClick={() =>
writeVideoUrl(video.id)}
      >
       <div className="items-center
flex space-x-2">
        <DynamicIcon name="link" />
        <span children={t('copyLink')}
/>
       </div>
      </ContextMenuItem>
      <ContextMenuItem
       className="flex items-center
justify-between"
       onClick={() =>
        writeVideoUrl(video.id,
Math.floor(videoState.currentTime))
       }
      >
       <div className="items-center
flex space-x-2">
        <DynamicIcon name="link" />
        <span
children={t('copyLinkFromCurrTime')}
/>
       </div>
      </ContextMenuItem>
     </ContextMenuContent>
    </ContextMenu>

    <div className="absolute top-1/2
left-1/2 transform -translate-x-1/2 -
translate-y-1/2 flex flex-col gap-y-
3">
     {videoState.showLoadingAnimation
&& !videoState.isLoading && (
      <DynamicIcon
       name={
        videoRef.current &&
videoRef.current.paused
         ? 'pause-circle'
         : 'play-circle'
       }
       className="animate-ping delay-
100 duration-1000 transition-all size-
14 bg-black/60 rounded-full"
      />
     )}
     {videoState.isLoading &&
!videoState.isDisabled && (
      <DynamicIcon
       name="loader-2"
       className="animate-spin
transition-all size-14 bg-black/60
rounded-full"
      />
     )}
     {videoState.isDisabled &&
video.status !== 'Preparing' && (
      <div
children={t('videoNotAllowedNow')} />
     )}

     {videoState.isDisabled &&
video.status === 'Preparing' && (
      <div
children={t('videoInProcessing')} />
     )}
    </div>

    <div
     className={`absolute transition-
all duration-300 bottom-0 w-full ${
      !videoState.showNavigationMenu
&& !videoState.isDisabled && hls ?
'opacity-100' : 'opacity-0'}`}
    >
     <Progress
value={(videoState.currentTime /
videoState.duration) * 100}
          className="h-1.5 w-full
rounded-b-lg rounded-t-none" />
    </div>

    <div
     className={`absolute transition-
all duration-200 bottom-0 w-full bg-
background/60 text-secondary-
foreground p-3 flex flex-col gap-y-3.5
${
      videoState.showNavigationMenu &&
!videoState.isDisabled && hls
       ? 'opacity-100'
       : 'opacity-0'
     }`}
    >
     <SliderPrimitive.Root
      defaultValue={[0]}
      min={0}
      max={videoState.duration}
      value={[videoState.currentTime]}
      step={0.5}
      onValueChange={event =>
onTimeUpdateHandler(event[0])}
      className="relative flex w-full
touch-none select-none items-center
group py-1"
     >
      <SliderPrimitive.Track
       className="relative w-full grow
overflow-hidden rounded-full bg-card
flex items-center justify-center h-1
hover:cursor-pointer">
       <div className="absolute h-1 w-
full">
```

```
        <div
         className="absolute h-1 bg-
primary"
         style={{
          width:
`${(videoState.currentTime /
videoState.duration) * 100}%`,
          left: 0
         }}
        />
        <div
         className="absolute h-1 bg-
input"
         style={{
          width:
`${(videoState.bufferedCount /
videoState.duration) * 100}%`,
          left:
`${(videoState.currentTime /
videoState.duration) * 100}%`
         }}
        />
       </div>
      </SliderPrimitive.Track>
      <SliderPrimitive.Thumb
       className="block size-2.5
group-hover:size-4 transition-all
duration-100 rounded-full border-2
border-primary bg-background ring-
offset-background focus-
visible:outline-none focus-
visible:ring-2 focus-visible:ring-ring
focus-visible:ring-offset-2" />
     </SliderPrimitive.Root>

     <TooltipProvider>
      <div className="flex justify-
between items-center">
       <div className="flex items-
center space-x-1.5">
        {video.prevId && (
         <Tooltip>
          <TooltipTrigger asChild>
           <Link
            href={getVideoUrl(
             video.prevId,
             undefined,
             query?.listId ?
(query.listId as string) : undefined,
             true
            )}
           >
            <DynamicIcon
             name="chevron-right"
             className="rotate-180"
            />
           </Link>
          </TooltipTrigger>
          <TooltipContent
children={t('prevVideo')} />
         </Tooltip>
        )}

        <Tooltip>
         <TooltipTrigger
          onClick={() => togglePlay()}
          children={
           <DynamicIcon
            name={
             videoRef.current &&
videoRef.current.paused
              ? 'play'
              : 'pause'
            }
           />
          }
         />
         <TooltipContent
          children={t(
           videoRef.current &&
videoRef.current.paused
            ? 'play'
            : 'pause'
          )}
         />
        </Tooltip>
        {video.nextId && (
         <Tooltip>
          <TooltipTrigger asChild>
           <Link
            href={getVideoUrl(
             video.nextId,
             undefined,
             query?.listId ?
(query.listId as string) : undefined,
             true
            )}
           >
            <DynamicIcon
name="chevron-right" />
           </Link>
          </TooltipTrigger>
          <TooltipContent
children={t('nextVideo')} />
         </Tooltip>
        )}

        <div className="group/volume
flex items-center gap-x-1.5">
         <Tooltip>
          <TooltipTrigger
           onClick={toggleMute}
           children={
            <DynamicIcon
             name={
              videoState.volume === 0
               ? 'volume-x'
               : videoState.volume <
0.4
                ? 'volume'
                : videoState.volume <
0.7
                 ? 'volume-1'
                 : videoState.volume <
1 ||
                  videoState.volume ===
```

```
1
                  ? 'volume-2'
                  : 'volume-x'
                }
               />
              }
             />
             <TooltipContent
              children={t(
               videoState.volume === 0 ?
'turnOnVolume' : 'turnOffVolume'
              )}
             />
            </Tooltip>
            <div
             className="transition-all
duration-200 ease-linear opacity-100
md:opacity-0 w-14 md:w-0 group-
hover/volume:opacity-100 group-
hover/volume:w-14">
             <Slider
              defaultValue={[0.5]}
              min={0}
              max={1}
              step={0.01}
              value={[videoState.volume]}
              thumbClassName="size-3.5"
              rangeClassName="h-0.5"
              trackClassName="flex items-
center justify-center h-0.5"
              className="w-16 group-
hover/volume:flex pr-2"
              onValueChange={event =>
changeVolume(event[0])}
             />
            </div>
            <span
             className="ml-1 group-
hover/volume:ml-1.5"

children={`${formatDuration(videoState
.currentTime)} /
${formatDuration(videoState.duration)}
`}
            />
           </div>
          </div>
          <div className="flex items-
center space-x-3">
           {video.nextId && (
            <Tooltip>
             <TooltipTrigger
              asChild
              className="flex items-
center"
              children={
               <Switch
                className={
                 videoState.autoPlayNext
? 'bg-primary' : 'bg-input'
                }

checked={videoState.autoPlayNext}
onCheckedChange={toggleAutoPlayNext}
               />
              }
             />
             <TooltipContent
              children={`${t('autoPlay')}
(${t(
               videoState.autoPlayNext ?
'allowed' : 'denied'
              )})`}
             />
            </Tooltip>
           )}

           <Tooltip>
            <TooltipTrigger
             className="flex items-center
hiddenOnMobile"
             asChild
             children={
              <button
               onClick={() =>
setCinemaMode(p => !p)}
               children={
                <DynamicIcon
                 name={cinemaMode ?
'monitor-x' : 'monitor-dot'}
                />
               }
              />
             }
            />
            <TooltipContent

children={`${t('cinemaMode')} (${t(
             cinemaMode ? 'allowed' :
'denied'
            )})`}
           />
          </Tooltip>

          <HoverCard open={menuOpen}>
           <HoverCardTrigger onClick={()
=> setMenuOpen(p => !p)}>
            <Tooltip>
             <TooltipTrigger>
              <DynamicIcon
name="settings" className="mt-1" />
             </TooltipTrigger>

<TooltipContent>{t('settings')}</Toolt
ipContent>
            </Tooltip>
           </HoverCardTrigger>

           <HoverCardContent
            side={isOpen ? 'bottom' :
'top'}
            align="end"
            className="flex flex-col
gap-y-2"
           >
```

```
          <Tooltip>
           <TooltipTrigger>
            <div className="flex
items-center space-x-2">
             <DynamicIcon name="gauge"
className="h-4 w-4" />
             <div
children={`${t('speed')}:
${videoState.speed}x`} />
            </div>
           </TooltipTrigger>
           <TooltipContent side="left"
align="end">
            <div
             className="flex flex-col
gap-y-2"
             children={[0.5, 1,
2].map((value, index) => (
              <Button
               key={index}
               size="sm"
               variant={
                value ===
videoState.speed
                 ? 'secondary'
                 : 'outline'
               }
               className="w-40"
               children={`${value}x`}
               onClick={() =>
changeSpeed(value)}
              />
             ))}
            />
            <TooltipArrow
className="w-3 h-2" />
           </TooltipContent>
          </Tooltip>

          <Tooltip>
           <TooltipTrigger>
            <div className="flex
items-center space-x-2">
             <DynamicIcon
name="settings-2" className="size-4"
/>
             <div>
{videoState?.levels?.[videoState.selec
tedLevel]
               ?.height
               ? `${t('quality')}:
${videoState?.levels?.[videoState.sele
ctedLevel]?.height}p`
               : `${t('quality')}:
${t('auto').toLowerCase()}`}
             </div>
            </div>
           </TooltipTrigger>
           <TooltipContent side="left"
align="end">
            <div className="flex flex-
col gap-y-2">
{videoState?.levels?.map((value,
index) => (
             <Button
              key={index}
              size="sm"
              variant={
               index ===
videoState.selectedLevel
                ? 'secondary'
                : 'outline'
              }
              className="w-40"

children={`${value.height}p`}
              onClick={() =>
changeQuality(index)}
             />
            ))}
            <Button
             key="auto"
             size="sm"
             variant={
              -1 ===
videoState.selectedLevel
               ? 'secondary'
               : 'outline'
             }
             className="w-40"
             children={t('auto')}
             onClick={() =>
changeQuality(-1)}
            />
           </div>
           <TooltipArrow
className="w-3 h-2" />
          </TooltipContent>
         </Tooltip>
         <HoverCardArrow
className="w-3 h-2" />
        </HoverCardContent>
       </HoverCard>
       <Tooltip>
        <TooltipTrigger
         onClick={toggleFullScreen}
         className="hover:animate-
pulse"
         children={
          <DynamicIcon

name={videoState.isFullScreen ?
'shrink' : 'maximize-2'}
          />
         }
        />
        <TooltipContent
children={t('fullScreenMode')} />
       </Tooltip>
      </div>
     </div>
    </TooltipProvider>
   </div>
  </div>)}
export default VideoPlayer
```

UNICHECK
by Turnitin

Ім'я користувача:
Кафедра КІ

ID перевірки:
1016353979

Дата перевірки:
13.06.2024 03:51:44 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
13.06.2024 06:28:32 EEST

ID користувача:
100005591

Назва документа: Мазур_Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнт...

Кількість сторінок: 68 **Кількість слів:** 13276 **Кількість символів:** 107217 **Розмір файлу:** 3.90 MB **ID файлу:** 1016157929

2.94%
Схожість

Найбільша схожість: 0.8% з джерелом з Бібліотеки (ID файлу: 1016157925)

2.46% Джерела з Інтернету	186	Сторінка 70
1.87% Джерела з Бібліотеки	180	Сторінка 72

0% Цитат

Не знайдено жодних цитат

Посилання	1	Сторінка 72

0%
Вилучень

Немає вилучених джерел

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 1.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 13%**

ID: 130049 Назва: БКР Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнтська частина) Додано в БД: 2024-06-12 Автора: Б. О. Мазур Керівники: В.М. Грига Консультанти: Опоненти:	Документ		Сумарний збіг по Базі Даних	
	Символи	Лексеми	Символи	Лексеми
	94075	737	1089 (1%)	11 (1%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Мазур Богдан Олегович

Тема: Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнтська частина)

Спеціальність: 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 Кількість сторінок записки 66

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є розробка клієнтської частини програмно-технічної системи для обробки, зберігання та передачі відеоконтенту з клієнтської сторони.

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі кваліфікаційної роботи проведено огляд предметної області щодо розроблення надійної та ефективної системи потокової передачі відео, яка забезпечить високу якість відтворення та зручність використання.

У другому розділі кваліфікаційної роботи проведено вибір компонентів та інструментів для розробки платформи потокової передачі відео. Було обрано Next.js як основний фреймворк для розробки додатка, HLS.js для реалізації адаптивної потокової передачі відео та Tailwind CSS для стилізації інтерфейсу. Спроектовано структурну та функціональну схеми додатка.

У третьому розділі кваліфікаційної роботи виконано процес розробки програмної реалізації платформи, включаючи інтеграцію HLS.js для автоматичного регулювання якості відео відповідно до пропускної здатності мережі користувача. Також розроблено та стилізовано інтерфейс додатка за допомогою Tailwind CSS. Проведено тестування для забезпечення стабільної роботи та високої продуктивності.

4. Позитивною стороною роботи є використання сучасних технологій та інструментів, що забезпечують високу якість відтворення відео та гнучкість у розробці інтерфейсу. Використання HLS.js дозволяє автоматично регулювати якість відео, забезпечуючи безперервне відтворення навіть при зміні пропускної здатності мережі. Tailwind CSS дозволяє швидко та ефективно стилізувати інтерфейс, що сприяє покращенню користувацького досвіду.

5. Негативні сторони роботи: Однією з негативних сторін роботи є складність інтеграції системи в існуючу інфраструктуру, що може вимагати додаткових ресурсів та часу. Це може ускладнити впровадження для організацій з обмеженими технічними можливостями.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно з діючими стандартами оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: _____

9. Оцінка дипломної роботи: відмінно

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) _____

Жолуда М.В. к.ф.-м.н. доцент кафедри ІПЗ

"14" серпня 2024 р. _____ (підпис)

Завідувачу кафедри КІІС
д-р.техн.наук, проф. Говорущенко Т. О.

Мазура Богдана Олеговича
ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи КІ2-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

11 червня 2024 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Програмно-технічна система обробки, зберігання та передачі відеоконтенту (клієнтська частина)

Автор: Мазур Богдан Олегович

Спеціальність: 123– Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Грига Володимир Михайлович, д.т.н, професор.

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи;
2) усі запозичення фрагментарні, або мають належним чином оформлені посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з 10-40 джерелами на один фрагмент речення;

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості, складає 2.94% і адресується до 366 першоджерела, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ В. М. Грига

Гарант ОП _____ С. М. Лисенко

Завідувач кафедри КІІС _____ Т. О. Говорущенко