МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп'ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ НАВЧАЛЬНОЇ ВЕБ-ПЛАТФОРМИ З ПІДВИЩЕНИМИ ВИМОГАМИ ДО БЕЗПЕКИ АВТОРСЬКОГО КОНТЕНТУ

Здобувач освіти гр. ІН.м–01н Дмитро Чехута

Науковий керівник, Олена Проценко
доцент, кандидат фізико-математичних наук

Завідувач кафедри Анатолій Довбиш
професор, доктор технічних наук

Суми 2022

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Чехуті Дмитру Андрійовичу

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія проєктування навчальної веб-платформи з підвищеними вимогами до безпеки авторського контенту".*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____
___

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Інформаційний огляд; 2) Вибір програмних засобів; 3) Практична реалізація; 4

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____
_____
____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Інформаційний огляд		
2.	Вибір програмних засобів.		
3.	Практична реалізація		
4.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____

РЕФЕРАТ

Записка: 69 стр., 17 рис., 1 таблиці, 1 додаток, 14 літературних джерел.

Об’єкт дослідження — Інформаційна технологія проєктування навчальної веб-платформи з підвищеними вимогами до безпеки авторського контенту.

Мета роботи — розробка додатку за допомогою за допомогою React, Firebase, Redux для забезпечення навчальних процесів в онлайн-форматі.

Результати — розроблено навчальну платформу для взаємодії суб’єктів навчального процесу. Також цю систему доповнено системою підвищеного захисту авторського контенту. Додаток був реалізований за допомогою мови програмування JavaScript, сервісу Firebase, бібілотек React та Redux.

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ ПРОЄКТУВАННЯ НАВЧАЛЬНОЇ ВЕБ-ПЛАТФОРМИ З ПІДВИЩЕНИМИ ВИМОГАМИ ДО БЕЗПЕКИ АВТОРСЬКОГО КОНТЕНТУ, INFORMATION TECHNOLOGY DESIGN OF A LEARNING WEB PLATFORM WITH INCREASED REQUIREMENTS FOR THE SECURITY OF THE AUTHOR'S CONTENT

ЗМІСТ

ВСТУП

У сучасному світі дуже важливий процес освіти, самоосвіти та саморозвитку. Для забезпечення цих процесів сучасні інформаційні технології пропонують таке рішення як навчальні веб-платформи.

Навчальна онлайн платфрома – інформаційний простір, що об'єднує учасників процесу навчання, що дає можливість для віддаленої освіти, забезпечує доступ до методичних матеріалів та інформації, а також дозволяє здійснювати тестування для контролю рівня знань учнів.

Онлайн-навчання (e-learning, дистанційне навчання, електронне навчання) – це спосіб отримання нових знань за допомогою Інтернету в режимі реального часу. На даний момент індустрія e-learning одна з технологій, що швидко розвиваються у світі, у сфері освіти. Навчання через Інтернет чудово підходить для тих, хто живе у віддалених районах, а також для тих, хто через певні причини не може відвідувати очну форму навчання.

Безперечними перевагами дистанційних курсів навчання в режимі онлайн також є:

- можливість для учня самостійно вибудовувати графік навчання, а також визначати тривалість занять.

- Вільний вибір. Учень обирає будь-який з доступних курсів навчання, а також самостійно планує час, місце та тривалість занять.

- Доступність. Незалежно від географічного розташування та часу учень має доступ до освітнього ресурсу та матеріалів курсу.

- Технологічність - використання в освітньому процесі новітніх досягнень інформаційних та телекомунікаційних технологій.

Популярність електронних освітніх платформ з інтерактивними можливостями навчання зростає, особливо за умов, продиктованих сучасною реальністю. Перехід на дистанційне навчання вимагає впровадження у процес сучасних методів навчання з використанням не тільки комп'ютерів, а й іншої сучасної техніки.

Найбільш популярні платформи для навчання здебільшого націлені на підвищення кваліфікації або здобуття нової професії, зараз стали з'являтися платформи, які можна використовувати і для дистанційного навчання дітей та підлітків. Основні напрямки курсів навчання, вибудованих на таких платформах: програмування, маркетинг, дизайн, тестування ПЗ, широке поширення набули всілякі онлайн-курси з іноземних мов: англійської, французької, іспанської тощо.

Всі платформи поєднують завдання, які вони дозволяють вирішувати, різниця лише в інтерфейсі самої платформи, тарифних планах, інформаційному наповненні та виборі методик викладання.

Семе тому актуальною є розробка навчальної веб-платформи, що дозволять не лише самостійно навчатися, а й дозволять взаємодіяти суб’єктам навчального процесу, роблячи навчальний процес більш ефективним та продуктивним. В той же час сучасна розробка потребує більш серйозного, кваліфікованого та якісного підходу до безпеки конфедеційної інформаці, коненту, функціональності тощо. Що створю нові виклики до контролю обмеженнями при розробці навчальних систем.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Аналіз предметної області

Інформаційні технології в освіті та навчанні з кожним днем стають все більш популярними. Інтерес до них підігріває постійне вдосконалення цифрового представлення інформації та спрощення доступу до будь-яких даних із застосуванням комп'ютерних мереж.

Сьогодні онлайн-навчання все частіше розглядається не просто як зручна форма підвищення кваліфікації, а як серйозна альтернатива традиційній освіті, що дозволяє отримати знання. Доступ до навчального курсу через Інтернет дає можливість у комфортних умовах та у зручний час вивчати матеріал, а значить – і краще його засвоювати.

Ще одним плюсом онлайн-освіти є чітко розроблена програма. Користувач вже на початку курсу розуміє, які етапи навчання йому доведеться пройти і які знання в результаті він отримає. Підтвердженням цього можуть бути різного роду дипломи, сертифікати та атестати.

Головна задача дистанційної освіти – проводити навчання, не маючи змоги безпосереднього контактувати з учнями. До сьогоденної освіти висуваються високенні вимоги, через вибуховий ритм сучасного життя та революційні технологічні можливості [5]:

Головні принципи, на основі котрих відбувається створення системи дистанційного навчання:

1. Доступність – наразі кожна сучасна людина стикається з хронічним браком часу, тому в якості альтернативи очної та заочної освіти виступає дистанційна освіта. При цьому освітній процес може розпочинатися в будь-який момент – при наявності часу; тривалість також залежить від наявності сил та часу в кожного конкретного учня; можливі паузи зупинки та продовження навчання. Також повинна бути врахована суб’єктивна

особливість кожного учня: його початковий рівень знань, особливості засвоєння матеріалу тощо. Колосальний прорив у даній галузі було здійснено завдяки результатам науково-технічного прогресу: появі персональних комп’ютерів, інтернету, мобільного інтернету та бездротових локальних мереж;

2. Принципово оновлені форми подання та організації інформації – це дозволяє забезпечити максимально високий рівень її сприйняття і засвоєння. Головні принципи[5]:

- використовувати по максимум різноманітні способи представлення інформації: текст, графічний контент, відео, звукові супроводи, анімації та все інше, що відноситься до поняття мультимедіа;
- наявність великого числа довідкової інформації, яка матиме супроводжуючий формат, тобто користувач зможе сам вирішувати – враховувати її чи ні – але довідкова інформація йому надається. Це може бути і суміжна з певним питанням інформація, і та, що безпосередньо викликає зацікавленість студента.
- доступність зворотного зв'язку від кураторів та інших учнів.

Наразі світова практика визнає найефективнішим способом організації різнопланової інформації – використання гіпермедійної технології. Найяскравішим прикладом її використання може виступити надпопулярна всесвітня мережа - World Wide Web. В контексті навчального процесу її можна розглядати в якості інтелектуальної надбудови над банком інформації глобальної мережі Інтернет.

3. Оцінювання знань в дистанційних формах освіти досить сильно ускладнюється. Під час класичного складання заліків та іспитів відбувається безпосередній контакт з викладачем, а під час отримання дистанційної освіти, це, як правило, неможливо. Тому в даних випадках застосовуються інші методики сертифікації знань. Найпоширеніша практика базується на проведенні тестувань.

Фактично, віддалену освіту можна порівняти з заочними формами навчання, а недостатність очного спілкування між викладачами та студентами компенсується за рахунок використання телекомунікаційних систем і технологій. Якщо надати більш розширене визначення для дистанційної освіти, можна сказали, що вона являє собою сукупність послуг освіти, надання яких відбувається з використанням спеціалізованих інформаційних освітніх середовищ, які базуються на принципах обміну інформацією для навчання віддалено. Тобто, інформаційно-освітні середовища – це система засобів прийому та передачі інформації, протоколів взаємодії, апаратного, програмного, організаційного, методичного забезпечення, яке отримує стуенд у вигляді дистанційних курсів.

Для того, щоб створити систему віддаленого навчання у освітньому закладі, необхідно виконати низку робіт, пов'язаних з кардинальною реорганізацією таких сфер діяльності, як адміністративна, методична та навчальна. Для виконання цих робіт необхідно[5]:

- Попередньо розробити концептуальні основи створення центру віддаленого навчання, враховуючи при цьому специфіку діяльності того чи іншого учбового закладу. Також необхідно врахувати існуючі концепції віддаленої освіт в України, чинну нормативну базу та приклади ефективної роботи систем віддаленого навчання. Це стосується і класичних навчальних закладів, і самостійних освітніх центрів різних мереж;
- Розробити кадрову структуру центру дистанційної освіти, вимоги до технічного забезпечення. Створювати бізнес-план та визначати умови економічної доцільності роботи необхідно враховуючи галузеву специфіку;
- Структурувати навчальні курси відповідно до модульного принципу, який дає змогу зручно поєднувати пакети навчально-методичних матеріалів відповідно до спеціальностей та спеціалізацій навчальних закладів, різних

рівнів базової освіти, професійної підготовки, додаткової освіти та можливостей підвищити кваліфікацію;

- Розробити комплексний план для підготовки навчально-методичного матеріалу для різних спеціальностей та спеціалізацій навчального закладу, який буде використовуватись під час віддаленого навчання;
- Організувати навчання для творців навчальних курсів – як використовувати інструментальні системи створення навчальних курсів для віддаленої освіти;
- Підготувати та провести апробацію навчально-методичних матеріалів, включно з електронними підручниками, системами для проведення тестувань, які будуть використовуватись для навчання і викладачів, і студентів.

Для того, щоб у навчальному закладі система віддаленої освіти функціонувала ефективно, обов’язково повинна бути наявною потужна телекомунікаційна база зі швидкісними каналами виходу до всесвітньої мережі. Без жодних сумнівів, навчальний центр повинен бути забезпечений комп’ютерною та телекомунікаційною базою на високому технічному та експлуатаційному рівні.

Таким чином, щоб створити систему дистанційного навчання, підготувати та організувати роботу центру віддаленої освіти в навчальних закладах – необхідно виконати комплексу роботу, яка буде стосуватися сфери навчально-методичного забезпечення, організаційних питань навчального процесу, питань забезпечення потужних комп'ютерної та телекомунікаційної систем, розробки сучасної зручної багатофункціональної платформи для навчання, та проведення підготовки викладачів.

1.2 Аналіз вимог до безпеки контенту в навчальних платформах

У поняття інформаційної безпеки освітньої установи входить система заходів, спрямована на захист інформаційного простору та персональних даних від випадкового або навмисного проникнення з метою розкрадання будь-яких даних або внесення змін у конфігурацію системи. [14]

У складі масивів інформації, що охороняється законом, які знаходиться в розпорядженні освітнього установи, можна виділити три групи:

- персональні відомості, що стосуються учнів та викладачів, оцифровані архіви;
- ноу-хау освітнього процесу, що носять характер інтелектуальної власності та захищені законом;
- структурована навчальна інформація, що забезпечує освітній процес (бібліотеки, бази даних, що навчають програми).

Всі ці відомості не тільки можуть стати об'єктом розкрадання. Умисне проникнення в них може порушити безпеку оцифрованих книг, знищити сховища знань, внести зміни до коду програм, що використовуються для навчання.

Обов'язками осіб, відповідальних за захист інформації, має стати збереження даних у цілісності та недоторканності та забезпечення їх:

- доступності в будь -яке час для будь-якого авторизованого користувача;
- захисту від будь-якої втрати або внесення несанкціонованих змін;
- конфіденційності, недоступності для третіх осіб.

Особливістю загроз стає не тільки можливість розкрадання відомостей або пошкодження масивів якимись свідомо діючими хакерськими угрупованнями, але й сама діяльність підлітків, навмисно або помилково здібних пошкодити комп'ютерне обладнання або внести вірус. Виділяються

чотири групи об'єктів, які можуть піддатися навмисному або ненавмисному впливу: [14]

- комп'ютерна техніка та інші апаратні засоби, які можуть бути пошкоджені внаслідок механічного впливу, вірусів, з інших причин;
- програми, що використовуються для забезпечення працездатності системи або в освітньому процесі, які можуть постраждати від вірусів або хакерських атак;
- дані, що зберігаються як на жорстких дисках, так і на окремих носіях;
- сам персонал, який відповідає за працездатність IT-систем;
- учні, схильні до зовнішнього агресивного інформаційного впливу.

Загрози, спрямовані на пошкодження будь-якого з компонентів системи, можуть носити як випадковий, так і усвідомлений навмисний характер. Серед загроз, що не залежать від наміру персоналу, учнів або третіх осіб, можна назвати[14]:

- будь-які аварійні ситуації, наприклад, відключення електроенергії або затоплення;
- помилки персоналу;
- збої у роботі програмного забезпечення;
- вихід техніки з ладу;
- проблеми у роботі систем зв'язку.

Всі ці загрози інформаційної безпеки носять тимчасовий характер, передбачувані та легко усуваються діями співробітників та спеціальних служб.

Цілеспрямовані загрози інформаційної безпеки носять більш небезпечний характер і в більшості випадків не можуть бути передбачені. Їх винуватцями можуть виявитися учні, службовці, конкуренти, треті. Для підриву інформаційної безпеки такі особи повинні мати високу кваліфікацію щодо принципів роботи комп'ютерних систем та програм. Найбільшій небезпеці піддаються комп'ютерні мережі, компоненти яких

розташовані окремо один від одного у просторі. Порушення зв'язку між компонентами системи може призвести до повного підриву її працездатності. [14]

Важливий проблемою може стати порушення авторських прав, навмисне розкрадання чужих розробок. [14]

З погляду проникнення в периметр інформаційної безпеки та для вчинення розкрадання інформації або створення порушення у роботі систем, необхідний несанкціонований доступ. [14]

Можна виділити кілька видів несанкціонованого доступу:

- Людський - інформація може бути викрадена шляхом копіювання на тимчасові носії, переправлена електронною поштою. Крім того, за наявності доступу до сервера зміни до баз даних можуть бути внесені вручну;
- Програмний - для розкрадання відомостей використовуються спеціальні програми, які забезпечують копіювання паролів, копіювання та перехоплення інформації, перенаправлення трафіку, дешифрування, внесення змін до роботи інших програм;
- Апаратний - він пов'язаний або з використанням спеціальних технічних засобів, або з перехопленням електромагнітного випромінювання різними каналами, включаючи телефонні.

Боротьба з різними видами атак на інформаційну безпеку повинна вестись на декількох рівнях, причому робота повинна мати комплексний характер. Існує низка методичних розробок, що дозволять побудувати захист освітнього закладу на необхідному рівні. Основні засоби для забезпечення інформаційної безпеки [14]:

1. Нормативно-правовий - захист інформації спирається на чинні у цій сфері закони, що визначають окремі її масиви як захист. Вони виділяють інформацію, яка повинна бути недоступна третім особам з різних причин (конфіденційна інформація, персональні дані, комерційна, службова або професійна таємниця). Крім законів необхідно виділити діючі у цій сфері

ДСТУ, що визначають порядок захисту даних, та застосовувані з цією метою методики та апаратні засоби.

2. Адміністративно-організаційний - цей комплекс заходів повністю побудований на створенні внутрішніх правил та регламентів, що визначають порядок роботи з інформацією та її носіями. Це внутрішні методики, присвячені інформаційній безпеці, посадові інструкції, переліки відомостей, що не підлягають передачі. Крім того, повинен бути визначений порядок доступу учнів до мережі Інтернет у комп'ютерних класах, та введена заборона на користування власними носіями інформації;

3. Технічний - комплексна система захисту всього периметра комп'ютерної мережі повинна забезпечуватися спеціалізованими програмними продуктами. Обов'язкове копіювання значущою інформації на диски комп'ютерів, які не мають доступу до мережі Інтернет. Обов'язково не тільки встановлення паролів, але їх регулярна заміна та інші інженерно-технічні заходи, які забезпечують конфіденційність, цілісність та доступність інформації;

4. Освітній – інтегроване систематичне навчання інформаційній безпеці в закладах освіти, а також підвищення кваліфікації для працівників органів державної влади та місцевого самоврядування, які працюють з інформацією.

Виходячи з вищесказаного, можна зробити висновки, що забезпечення безпеки всієї інформації освітніх установ – вкрай важливе завдання, всі заходи щодо забезпечення безпеки повинні застосовуватися в комплексі та враховувати різні можливі аспекти інформаційної безпеки.

1.3 Веб-додатки: види, архітектура та принципи роботи

Веб-програма — це програма, одна частина якої завантажується в браузер і взаємодіє з користувачем (візуально-інтерфейсна частина), а інша знаходиться на веб-сервері та виконує запити, що надходять від першої, а потім повертає відповідь. Частина, яка завантажується в браузер і з якою

взаємодіє користувач, називається частиною клієнта (фронтенд). На веб-сервері знаходиться серверна частина веб-програми (бекенд). [11]

Незважаючи на те, що і веб-сайтом, і веб-програмою користуються за допомогою браузера, між ними є суттєві відмінності.

Веб-сайт – це сукупність веб-сторінок, найчастіше інформаційного характеру. Він може містити контент у вигляді тексту, зображень, аудіо або відео тощо. Веб-сайти видають користувачеві готові HTML-сторінки, доступні для перегляду. Взаємодія із ними обмежена. Найчастіше можна лише скористатися пошуком або підписатися на новини. Аутентифікація не є обов'язковою. Сайт компанії – характерний приклад веб-сайту. Також такі сайти часто називають статичними. [11]

Щоб користуватися можливостями веб-додатку, потрібно пройти авторизацію, інакше будь-хто зможе отримати доступ до даних користувача та налаштувань. Прикладом веб-додатку може виступати соціальна мережа.

Основні відмінності між веб-додатком та веб-сайтом представлені в таблиці 1.

Веб сайт	**Веб-додаток**
Дозволяє переглядати дані	Дозволяє маніпулювати даними
Можна користуватися без автентифікації	Потрібна автентифікація
Видає заздалегідь підготовлені HTML-сторінки в основному зі статичними файлами	Фрагменти HTML-сторінки генеруються та оновлюються «на льоту»
Простіше у розробці; менше налаштувань для відвідувача	Потребує розробки; дає більше налаштувань для користувача. Це породжує складність, зворотний бік чого — потенційні помилки.

Таблиця 1 – Основні відмінності між веб-додатком та веб-сайтом

Незважаючи на всі відмінності, в деяких випадках межі між веб-сайтами і веб-додатками стираються - веб-сайти можуть включати в себе веб-додатки або мати деякі їх характеристики. [11]

Прикладами веб-додатків є:

- інтернет-пошта (наприклад, Gmail);
- текстові редактори (наприклад, Google Документи);
- соціальні мережі (наприклад, Facebook);
- магазини електронної комерції (наприклад, Ebay);
- хмарні сховища (наприклад, Dropbox);
- онлайн-нотатки (наприклад, Evernote);
- системи управління проектами (наприклад, Trello).

Веб-додаток має багато переваг:

- Не потребує встановлення на жорсткий диск і тому не займає багато простору;
- Не вимагає оновлення, тому що оновлюється централізовано;
- Ви можете користуватися з будь-якого пристрою, на якому є веб-браузер;
- Не залежить від платформи та операційної системи (ОС): якщо веб-програма сумісна з браузером, вона працює;
- Розробнику не потрібно створювати клієнтські програми для різних ОС, тому що використовується браузер.

Веб-програми можуть розроблятися та використовуватися з різними цілями. Вони дозволяють обмінюватися інформацією та проводити транзакції, продавати та купувати товари та послуги онлайн, спільно працювати над проектами, створювати текстові файли, електронні таблиці та презентації, відкривати доступ до них іншим користувачам. [11]

За допомогою веб-додатків можна працювати з нотатками, списками завдань, керувати файлами в хмарі, перетворювати величини з одних одиниць виміру на інші, перекладати тексти, створювати парсери і так далі - вони застосовуються в багатьох різних сферах.

Залежно від розподілу навантаження між клієнтською та серверною частиною, можна виділити кілька типів архітектури: від переважно серверних до переважно клієнтських. Окремо стоять прогресивні веб-застосунки, яким доступні деякі можливості десктопних додатків. [11]

У серверних веб-додатках все навантаження покладено на серверну сторону. Програма приймає запит, визначає яку сторінку потрібно вивести, та повертає відповідну HTML-сторінку. Вона може бути як статичною, так і динамічною. [11]

Для формування відповіді бекенд може звертатися до бази даних за необхідною інформацією для заповнення шаблону сторінки.

Веб-програми з використанням AJAX. При першому запиті на сторінку передається HTML-код каркаса. Код JavaScript асинхронно підвантажує решту фрагментів сторінки і може «на льоту» надсилати запити на сервер та обробляти його відповіді у форматі XML (eXtended Markup Language) або JSON (JavaScript Object Notation). Ця технологія називається "асинхронний JavaScript та XML" (Asynchronous JavaScript And XML, AJAX). [4]

Клієнтські програми. Все навантаження покладено на сторону клієнта. Сервер лише доставляє HTML-код із посиланнями на стилі та сценарії JavaScript, а ці сценарії забезпечують логіку, відображення та підвантажують потрібний контент.

Вся взаємодія з користувачем відбувається на одній сторінці, тому такі програми називають односторінковими (single page applications, SPA). Користувач виконує деякі дії, надсилає запит та отримує відповідь без перезавантаження сторінки. [11]

Для створення односторінкових веб-застосунків використовуються такі фреймворки, як, наприклад, Ember.js, Angular, React, Backbone.js і Vue.js.

Прогресивні веб-додатки (англ. progressive web application, PWA) — це веб-програми, розроблені за допомогою певних спеціальних технологій і стандартних шаблонів, що дозволяє їм користуватися перевагами десктопних та веб-додатків. [11]

Прогресивні веб-застосунки можуть зберігати дані на стороні клієнта, тому ними можна користуватися без підключення до інтернету і робота з даними ведеться швидше.

Вони мають такі характеристики:

- доступність для пошуку: контент можна знайти через пошукові системи;
- можливість встановлення: програми доступні з домашнього екрана пристрою або засобу запуску програм;
- можливість створення посилань: можна передати посилання на прогресивний веб-додаток іншому користувачеві;
- незалежність від мережі: працюють без підключення до інтернету або за слабкого підключення;
- прогресивне удосконалення: доступні на базовому рівні у старих браузерах, а повний функціонал доступний у нових браузерах;
- повторне залучення: коли доступний новий контент, програма може надсилати повідомлення;
- адаптивний дизайн: доступні на будь-яких пристроях з екраном та браузером, у тому числі на мобільних телефонах, планшетах, ноутбуках, телевізорах, холодильниках тощо;
- безпека: з'єднання між користувачем, програмою та сервером захищене, і треті сторони не можуть отримати доступ до конфіденційних даних.

Принципи роботи веб-додатків

Веб-додатки складаються з серверної частини (back-end, бекенд) та клієнтської частини (front-end, фронтенд). Користувачі взаємодіють із клієнтською частиною через інтерфейс, що відображається у браузері (Chrome, Firefox, Safari, Edge та ін.). За командою користувача запит надсилається на сервер через Інтернет. На сервері його обробляє серверний код та повертає клієнту відповідь. [11]

У відповіді може бути як готова HTML-сторінка, так і шаблон сторінки або дані, наприклад, у форматі XML або JSON. Це залежить від вибраного типу рендерингу (формування) сторінки. Тобто, сторінка може відправлятися взагалі без змін (статична сторінка) або бекенд вносить до неї зміни, після чого відправляє її браузеру (динамічна сторінка). Рендеринг може здійснюватися або повністю на сервері, або у різних співвідношеннях розподілятися між сервером та клієнтом, або виконуватись лише клієнтом.

Сторінки перших веб-сайтів містили лише текст у форматі HTML. Згодом додалися зображення та таблиці, але веб-сторінки залишалися статичними у буквальному значенні слова. Згодом з'явилися технології, які дозволили надати їм динаміки.

Обробка статичних веб-сторінок. Клієнт надсилає серверу навігаційний запит. У відповідь на запит сервер передає клієнту статичну веб-сторінку без змін. Весь її контент (текст, зображення, контент тощо) щоразу виводиться однаково. Цей контент "жорстко" закодований у самій сторінці. [11]

Обробка динамічних сторінок

Бекенд. 1993 року з'явилася специфікація Common Gateway Interface (CGI). Це інтерфейс, який використовується програмою для зв'язку із веб-сервером. Така програма називається шлюзом. Шлюз може бути написаний будь-якою мовою програмування, яка використовує стандартне введення-

виведення: C/C++, Fortran, PERL, TCL, будь-яка оболонка Unix (shell), Visual Basic, AppleScript. Більшість сценаріїв було написано на Perl. [11]

Ці сценарії дозволяли використовувати той самий шаблон, щоб наповнювати його різним контентом. Таким чином, сторінки стали динамічно генеруватися на сервері у тексті сценарію. У той же час, з огляду на дії користувача в режимі реального часу сторінки залишалися статичними.

У 1995 році з виникненням JavaScript з'явилася можливість реагувати на дії користувача миттєво та відкривати спливаючі вікна. Веб-сторінки пожвавішали. У цьому ж році було створено мову PHP. Він дозволяв поєднувати HTML-код із логікою.

Вигода - не доводиться створювати кожну сторінку окремо. Один і той самий код виконує рендеринг будь-якої сторінки. Такий рендеринг називається серверним рендерингом (server-side rendering, SSR). Сервер обробляє запит, формує сторінку шаблону, а клієнт отримує готову повнофункціональну HTML-сторінку.

Зараз для написання коду бекенда використовується безліч мов та спеціальних фреймворків. Найпопулярніші - це Java (з використанням Java Servlet API), PHP + Laravel, Python + Django, Node.js, мови платформи .NET (C#, VB) + ASP.NET, Ruby + Ruby on Rails та Go. Вибір конкретної мови та фреймворку залежить від характеру розв'язуваних завдань.

Список продуктів з інформацією про них може зберігатися у базі даних (БД). Із нею взаємодіє серверний код. Він може читати дані з бази, додавати, змінювати чи видаляти їх. Як система управління базою даних використовуються MySQL, PostgreSQL, Memcached, MongoDB, Redis та інші. Для роботи з БД існує безліч бібліотек, орієнтованих різні серверні мови програмування.

Фронтенд. Інтерфейс веб-програми може передаватися браузеру не лише у вигляді цілої HTML-сторінки. Наприклад, в односторінковому

додатку це каркас: мета-теги, посилання на стилі та сценарії, а також елементи, зазвичай div, в які сценарій підставляє потрібний контент. [11]

Остаточний вигляд програма набуває після завантаження всього контенту та залучення всіх стилів. Але і після цього деякі меню та списки формуються динамічно при натисканні відповідних кнопок, а деякі оновлюються в режимі реального часу. Наприклад, коментар до публікації можна надіслати не залишаючи сторінку, і він з'явиться, як тільки буде встановлено, що він успішно доданий до сховища.

Фронтенд може містити інформаційні блоки та елементи керування. Наприклад, у Facebook інформаційні блоки – це публікації у стрічці, історії, рекомендації, а елементи управління – кнопки вкладок, меню, посилання, поля пошуку та введення контенту для публікації, коментарів тощо.

Якщо взяти, наприклад, Google Docs (див. рис. 1.1), то бачимо, в основному, такі елементи управління: меню, панель інструментів, панель структури, документ. Інформаційний блок тут – це довідка. Загалом це інтерактивний інтерфейс.

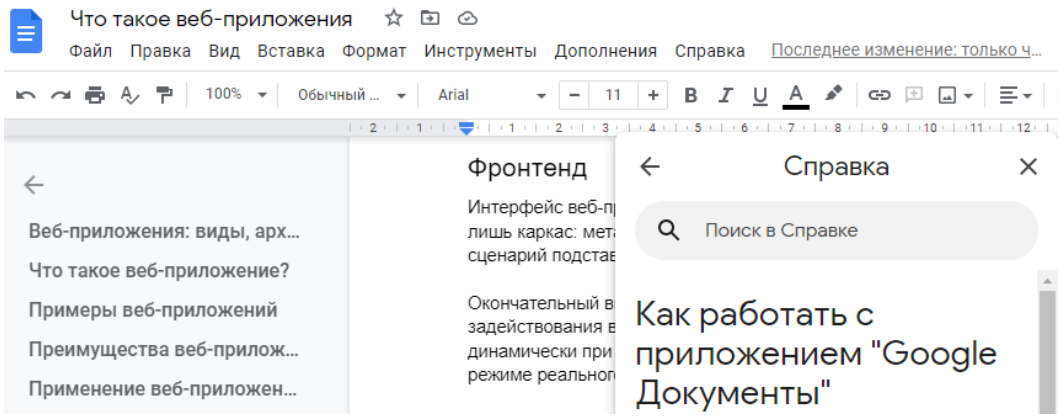

Рисунок 1.1 – Вигляд вікна веб-додатку Google Docs

Інтерактивність забезпечується за рахунок JavaScript-коду, який виконується у браузері. Для складних програм використовуються спеціальні бібліотеки, що спрощують написання коду. Наприклад, Google Docs та багато інших веб-додатків Google використовують бібліотеку Closure Library, Facebook — бібліотеку React. Бібліотека Redux застосовується для керування станом програми та часто використовується з React. Широко

використовуються такі бібліотеки для веб-додатків, як Angular, Ember.js, Express.js (у парі з Node.js), Vue.js.

Завдяки використанню цих бібліотек можна використовувати рендеринг клієнта (client-side rendering, CSR). В односторінковому додатку рендеринг, логіка та завантаження покладаються на сторону клієнта.

Веб-застосунки впевнено займають своє місце в інтернеті і продовжують еволюціонувати. Це пов'язано з зручністю їх застосування, а також готовністю до використання на мобільних пристроях. У свою чергу, розвиваються фреймворки для їхньої розробки (і з'являються нові). Враховуючи безліч зручних фреймворків як для бекенда, так і для фронтенду, немає необхідності винаходити складні рішення з нуля при розробці веб-додатка. Можна повністю сконцентруватись на поставлених задачах.

1.4 Постановка задачі

Метою роботи є створення навчальної веб-платформи, яка буде забезпечувати взаємодію між студентом та ментором шляхом реалізації механізмів навчального процесу, таких як створення задач, їх перевірка, їх оцінювання та діалогу між суб’єктами навчального процесу. А також запровадити систему підвищенного захисту, яка буде базуватися на системі контролю доступу до функціоналу та контенту.

Система дозволить менторам створювати свої задачі з будь-якою конфігураціє, а також редагувати їх. При створенні задачі можна буде активувати систему створення штрафі, наприклад, за зрив дедлайну або ж за якісь грубі порушення. Також прис створенні та редагуванні задач стане можливим групувати критерії задачі в категорії, наприклад, базовий рівень, продвинутий рівень, експертний рівень і т.д.

Система дозволить менторам та студентам робити первірки та самопервірки виконаних задач. Система надасть можливість перевіряючому під час перевірки додавати критерії штрафів або бонусів до вже

сформованої задачі. Для реалізації вищезазначеної системи сформульовані наступні задачі:

– Провести аналіз предметної області;

– Спроектувати систему та її частини;

– Розробити схеми функціональних можоивостей користувачів з різними ролями в додатку;

– Розробити моделі даних головних сутностей;

– Розробити систему контролю доступу, яка буде обмежувати доступ користувачів до функціональності чи контенту за певними умовами;

– Створити логіку функціонування додатку;

– Забезпечити взаємодію між клієнтом та сервером

2 ВИБІР ПРОГРАМНИХ ЗАСОБІВ

2.1 Вибір фреймворку React.js

Фреймворк - це каркас програми, який складається з набору бібліотек.

Бібліотека - код, який вбудовується у додаток та вирішує обмежений набір завдань залежно від того, які обов'язки на неї поклали розробники. Як правило, бібліотека — контейнер для набору класів та/або функцій, які розробники можуть використовувати у своєму додатку. [10]

Разом вони дають можливість користувачеві побудувати свою функціональність на певному фундаменті. Фреймворк задає структуру та підхід до архітектури програми, він диктує, як повинен писатися код.

Додаток може бути побудовано як тільки на одних бібліотеках, так і на одному фреймворку. Але частіше всього розробники використовують фреймворк та сторонні бібліотеки одночасно.

Фреймворки дозволяють створювати унікальний набір функцій і підходять для нетипових продуктів. JavaScript-фреймворки підвищують продуктивність розробника за рахунок того, що вже містять у собі певні правила, шаблони та набори функцій. [10]

Вибір того, який каркас взяти або з яких блоків побудувати новий проект, залежить від кругозору інструментів і поставлених задач.

Сучасні фреймворки достатньо функціональні та дозволяють розробникам реалізувати будь-яке завдання. Їх вибір досить широкий, але під час роботи над проектом були розглянуто 3 найпопулярніші - React, Vue та Angular.

У 2010 році Google випустив Angular і фреймворк зробив революцію, розробники його оцінили. Але продуктивність на той момент залишала бажати кращого. У 2016 році вийшов Angular2, написаний з нуля на TypeScript. Ця версія мала цілковито іншу архітектуру, більш низький поріг входження та якісну документацію з великою кількістю прикладів. Серед

відомих компаній, що використовують цей фреймворк - Google, Forbes, PayPal, Upwork та ін.

Майже паралельно з Angular розвивався React. Вперше його використали у новинній стрічці Facebook у 2011 році, але вихідний код відкрили лише у 2013 році. Поступово він набрав хорошу базу, велику кількість інструментів та широку підтримку ком'юніті. React використовується для розробки інтерфейсів у багатьох відомих компаніях: Facebook, Instagram, Netflix, BBC та ін.

2014 року з'явився Vue.JS. Біля його витоків стояв колишній співробітник Google . У фреймворку немає підтримки зі сторони великих компаній на відміну від React та Angular. Але це не завадило третій версії Vue розміщуватися у власному репозиторії GitHub і перейти на TypeScript. Це забезпечило їм більш чисту та зручну архітектуру вихідного коду, а також ряд інших змін. Vue використовується, в основному, китайськими гігантами - Alibaba, Tencent, Xiaomi, а також іншими великими компаніями.

Якщо звернутися до статистики числа завантажень (див. рис. 2.1), то React обирають частіше, ніж решту два фреймворки.

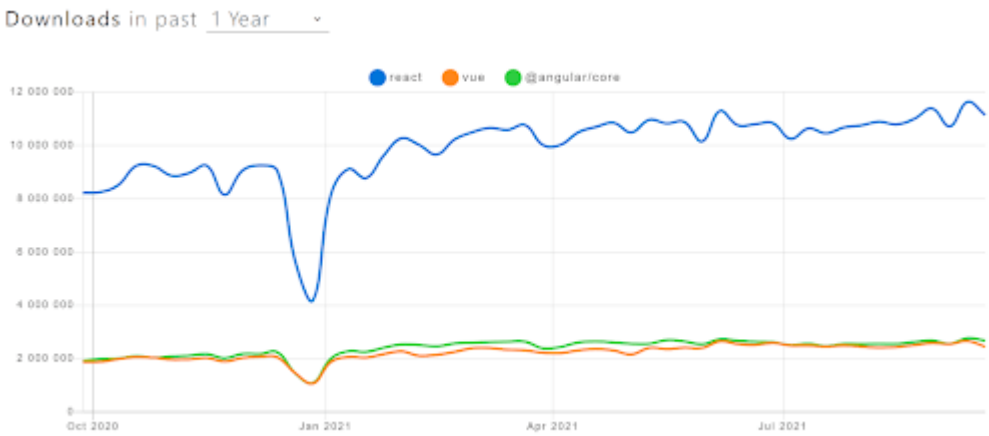

Рисунок 2.1 – Статистика числа завантажень різних фреймворків

Але за кількістю зірок (рейтинг затребуваності), які отримують репозиторії GitHub цих фреймворків, лідирує Vue. Хоча він і не набагато випереджає React. Рейтинг GitHub представлений на рис. 2.2.

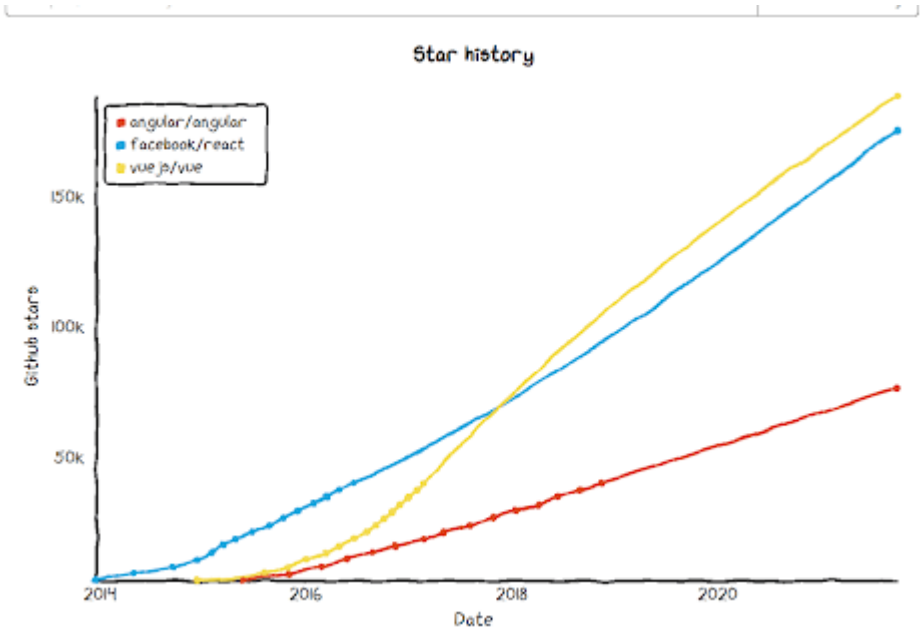

Рисунок 2.2 – Рейтинг затребуваності фреймоворків на веб-сервісі для розробників GitHub

Angular як фреймворк використовується для розробки великих корпоративних додатків, оскільки ряд архітектурних рішень йде з коробки, включаючи TypeScript. Це, з одного боку, може бути обмеженням, а з іншого – швидким рішенням стандартного кейсу. Також є механізми DI для зручності тестування та підміни залежностей. Angular як фреймворк вузькоспеціалізований. І поріг входження у розробку у ньому досить-таки високий. Крім знань JS-розробника, необхідно добре розуміти, як влаштований цей фреймворк. Можливо, саме тому він не користується особливою популярністю.

Vue підходить для середніх додатків та MVP. Висока швидкість на старті дозволяє зробити MVP за короткі терміни. Масштабованість програми Vue допомагає також динамічно розвивати проект. У версії 3.0 є повноцінна підтримка TypeScript. Легкість масштабування вища, ніж у Angular.

React, по своїй суті, є бібліотекою і має великий арсенал суміжних бібліотек. Він підходить для вирішення будь-яких задач. Його можна застосувати в MVP, невеликих та середніх проектах. Підтримка TypeScript

допоможе в enterprise-розробках. Також його можна вибрати для розробки сайтів з мінімальною бізнес-логікою на фронті, оскільки є бойлерплейт (шаблонний код).

Але при цьому у React немає «нав'язаних» рішень з коробки, тому потрібні хороший досвід розробки та знання як архітектурних патернів, так і роботи нативного JS.

Виходячи з того, що для роботи проектованої навчальної веб-платформи буде потрібна висока продуктивність, а реалізація всього передбаченого на платформі функціоналу потребуватиме широкого спектру інструментів та можливостей – було прийнято рішення використовувати фреймворк React.js.

React не відноситься до фреймворків у чистому вигляді. Це свого роду модифікована бібліотека, «заточена» під MVC (Model-View-Controller, де Модель відповідає за надання даних, Вид – відповідає за відображення даних Моделі користувачеві, а Контролер інтерпретує дії користувача та змушує Модель вносити зміни).

React — бібліотека для відображення даних, яка включає в себе Virtual DOM, JSX, ізоморфність та компонентний підхід. [9]

React забезпечує підвищену гнучкість завдяки використанню «компонентів» - коротких ізольованих ділянок коду, які допомагають розробникам створювати складну логіку та UI. React взаємодіє з HTML через virtual DOM - копію реального DOM-дерева елементів сторінки. У копії всі елементи представлені як об'єкти JavaScript. Ці елементи, разом із декларативним стилем програмування React та одностороннім зв'язуванням даних, спрощують та прискорюють розробку . [9]

Віртуальний DOM — об'єкт, в якому зберігається інформація про стан інтерфейсу. Якщо стан змынюэться, наприклад, выдбувається відправка форми або натискається кнопка, React робить розрахунок різниці між попереднім та новим станами. Далі бібліотека відображає новий стан.

Використання віртуального DOM дозволяє бібліотеці ефективно оновлювати реальний DOM.

React відрізняється від інших фреймворків та бібліотек тим, що не має вбудованого архітектурного шаблону. Він використовує компонентно-орієнтовану архітектуру . Користувальницькі інтерфейси React рендеруються компонентами, які працюють як функції та реагують на зміну даних. Таким чином, внутрішня архітектура - постійна взаємодія між станом компонентів та діями користувачів. [9]

Одна з головних переваг React – легка масштабованість. Оскільки програми React використовують виключно JavaScript, розробники можуть застосовувати традиційні методи організації коду. Можливість багаторазового використання компонентів підвищує масштабованість додатків на даному фреймворку .

Ще одна перевага фреймворку - повна сумісність між версіями. Можливе підключення до додатку бібліотек різних версій, оновлення застарілих з успадкуванням властивостей.

За допомогою React можливо створювати веб-програми, які змінюють відображення без перезавантаження сторінки. Завдяки цьому програми можуть з високою швидкістю реагувати на дії користувачів, такі як заповнені форми, застосовані фільтри, додані у кошик товари тощо.

React застосовують для відображення компонентів інтерфейсу користувача. Також бібліотека може повністю управляти фронтендом. У цьому випадку React використовують разом з бібліотеками для керування станом і роутингу, наприклад, Redux і React Router.

Одна з ключових переваг React - універсальність. Цю бібліотеку можна використовувати на сервері та на мобільних платформах за допомогою React Native. Це принцип Learn Once , Write Anywhere або «Навчіться один раз, пишіть де завгодно».

Ще одна важлива особливість бібліотеки - декларативність. Використовуючи фреймворк, розробники можуть описувати вигляд

компонентів інтерфейсу в різних станах. Така декларативність дозволяє суттєво скоротити код і зробити його зрозумілим.

React заснований на компонентах, це ще одна ключова особливість бібліотеки. Кожен компонент повертає частину користувальницького інтерфейсу зі своїм станом. Виконуючи об'єднання компонентів, розробник може створити завершений інтерфейс веб-програми.

Черговий плюс фреймворку — використання JSX. Це розширення синтаксису JavaScript, яке зручно використовувати для опису інтерфейсу. JSX схожий на HTML, проте це все-таки JavaScript. [11]

Таким чином, можна зробити висновок, що фреймворк React.js оптимально підходить для розробки проектованої у цій роботі навчальної веб-платформи. Використовуючи його, можна легко створювати складні великомасштабні динамічні додатки, будувати користувальницькі інтерфейси. React.js забезпечує високий рівень продуктивності, має величезний вибір гнучких бібліотек та інструментів. Саме тому він був обраний для реалізації практичної частини даної роботи.

2.2 Вибір середовища розробки WebStorm

WebStorm - це інтегрована середовище для розробки на JavaScript та пов'язаних з ним технологіях. WebStorm дозволяє автоматизувати рутинну роботу і легко справлятися з складними завданнями, роблячи розробку більш простою і зручною. [12]

IDE створено на базі платформи IntelliJ, розробленої JetBrains та випущеної під відкритої ліцензією.

У WebStorm є вбудована підтримка множини технологій : JavaScript, TypeScript, React, React Native, Electron, Vue, Angular, Node.js, HTML, CSS та інших. Можна відразу приступати до написання коду, не витрачаючи час на встановлення та налаштування плагінів.

У WebStorm наявний розумний редактор коду. IDE добре розуміє структуру проектів та допомагає з усіма аспектами написання коду:

- Автодоповнення коду - дозволяє писати код швидше, використовуючи пропоновані IDE на ходу ключові слова та символи. Усі підказки формуються з урахуванням контексту та типів, а також працюють для різних мов. Наприклад, можна побачити підказки для імен класів з CSS під час роботи з .js файлами. Приклад автодоповнення коду представлений на рис. 2.3;

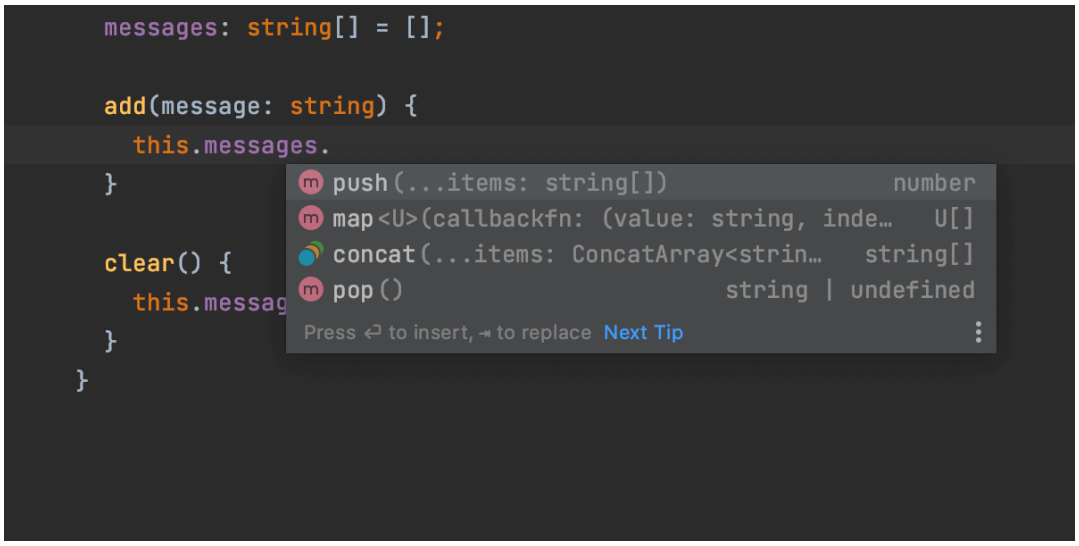

Рисунок – 2.3 – Приклад автодоповнення коду в IDE WebStorm

- Аналіз якості коду - WebStorm дозволяє з легкістю виявляти помилки у коді. IDE включає сотні інспекцій для всіх підтримуваних мов, а також перевірку правопису. Також вона інтегрується зі Stylelint, ESLint та іншими лінтерами - WebStorm запускає їх, поки пишеться код, і підсвічує помилки прямо у редакторі. IDE пропонує багато варіантів швидких виправлень;
- Безпечні рефакторинги - WebStorm включає потужні інструменти для рефакторингу коду по всій кодовий базі. Є можливість перейменовувати файли, папки та символи, отримувати компоненти, методи та змінні і не боятися, що ці дії приведуть до помилок. Редактор повідомляє про будь-які потенційні проблеми;
- Швидкий перегляд документації – якщо виникає необхідність подивитися документацію для символу, для цього можна не виходити з IDE.

Достатньо навести курсор миші на потрібний символ, і після натискання комбінації клавіш Ctrl+Q можна переглянути відповідну інформацію (див. рис. 2.4). Також WebStorm підказує параметри для методів та функцій, які викликаються;

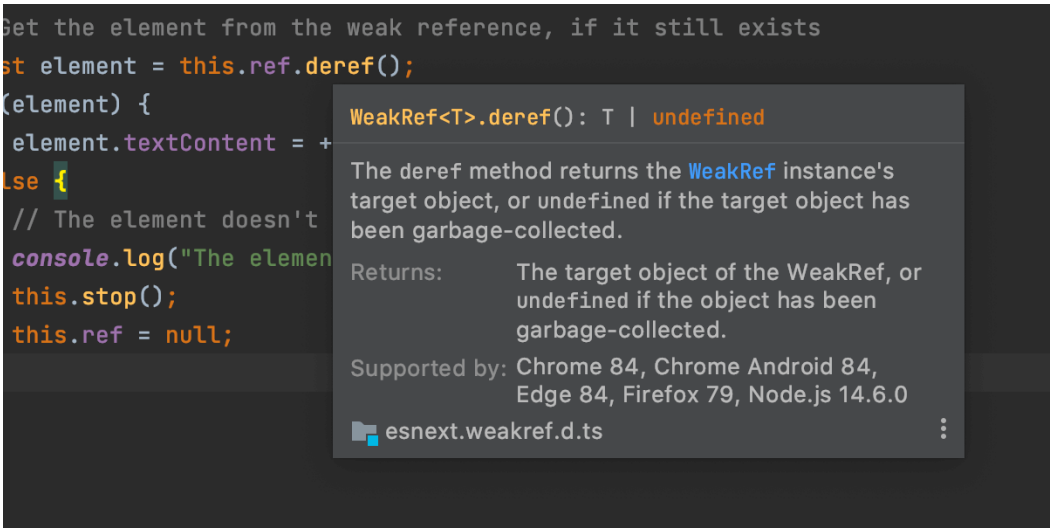

Рисунок 2.4 – Швидкий перегляд документації в IDE WebStorm

- Попередній перегляд HTML-файлів - у WebStorm є прев'ю для статичних HTML-файлів. Під час редагування HTML-коду або пов'язаних CSS та JavaScript-файлів, зміни зберігаються і прев'ю оновлюється автоматично.

Також доступні вбудовані інструменти для розробників. Один з головних плюсів IDE у тому, що вони об'єднують усі необхідні інструменти. WebStorm можна використовувати для налагодження та тестування клієнтського коду та програм на Node.js, а також для роботи з системою контролю версій. При розробці зручно користуватися: [12]

- Налагодженням JavaScript - запуск та налагодження коду клієнтських додатків та додатків на Node.js можна робити прямо у редакторі. Є можливість розставляти точки зупинки, виконувати програму покроково, додавати watches і т. д. – все це працює для самих різних типів додатків, включаючи JavaScript, TypeScript та Vue;
- Юніт-тестуванням – передбачена можливість писати, запускати та налагоджувати юніт-тести за допомогою Jest, Mocha, Karma, Protractor та

Cucumber.js. Переглядати результати тестів можна у вигляді дерева, що дозволяє переходити до вихідного коду тесту;

- Просунутою інтеграцією з VCS - інтерфейс WebStorm підтримує безліч повсякденних операцій: порівняння гілок, перегляд та вирішення конфліктів і т. д. IDE також дозволяє працювати з проектами, розміщеними на GitHub;
- Локальною історією змін – на випадок, якщо при розробці не було зроблено коміт або були випадково видалені кілька файлів, у WebStorm є вбудована локальна історія. Вона відстежує всі зміни в рамках проекту та дозволяє скасувати їх, навіть якщо проект поки що не підключений до системи контролю версій; [12]
- Вбудованим HTTP- клієнтом – є можливість тестувати веб-сервіси за допомогою вбудованого HTTP-клієнта - створювати, редагувати та виконувати HTTP- запити прямо в редакторі;
- Підтримкою лінтерів – доступна можливість інтеграції WebStorm з популярними лінтерами, такими як ESLint, Stylelint або TSLint. Це дозволить переглядати згенеровані ними попередження та помилки прямо в IDE і швидко виправляти проблемний код;
- Вбудованим терміналом - можливе використання вбудованого терміналу для роботи з будь-якою командною оболонкою прямо з IDE.

Ще одна перевага редактора - швидка навігація та пошук. За допомогою даної IDE можливо швидко переміщатися кодом незалежно від розмірів проекту. Доступний пошук файлів, класів або символів, є можливість переглядати всі результати в одному місці. [12]

Головні опції навігації та пошуку:

- Вспливаюче вікно Search Everywhere - допоможе знайти у WebStorm все що завгодно. Його можна використовувати для пошуку дій, файлів, класів або символів та перегляду всіх знайдених збігів;

- Дослідження коду - редактор дозволяє швидко перейти до оголошення символу і показує де він використовується в проекті. Також є можливість легко подивитися визначення символу, не переходячи до його оголошення;
- Навігація проектом – можна переглядати файли проекту, переходити до файлів та фрагментів коду, які недавно були відкриті або редагувалися — WebStorm запам'ятовує, з чим ведеться робота, і дозволяє швидко повернутися назад. Для навігації використовуються вкладки або поєднання клавіш. Вид вікна навігації представлений на рис. 2.5;

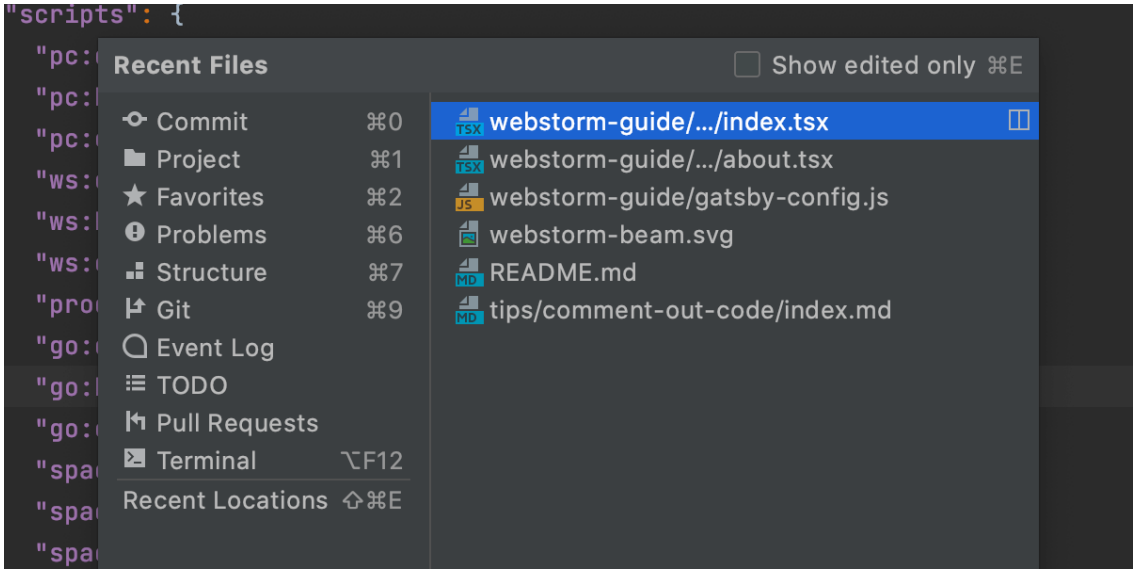

Рисунок. 2.5 – Вікно навігації у IDE WebStorm

- Пошук тексту - IDE дозволяє знаходити та замінювати текстові рядки як у певних файлах, так і у всьому проекті. Можливо звузити область пошуку до виділеного фрагмента коду або використовувати різні області проекту та фільтри.

Ефективна командна робота теж є важловою складовою редактора. WebStorm дозволяє програмувати разом із командою в режимі реального часу та спілкуватися з колегами прямо з IDE. Головні можливісті:

- Віддалена спільна розробка - WebStorm включає Code With Me - сервіс для спільної віддаленої розробки та парного програмування. Його

зручно використовувати, щоб разом працювати над кодом та спілкуватися з колегами, не залишаючи IDE;

- Можливості для розподілених команд – дозволяють створювати пул-реквести GitHub, об'єднувати їх з цільовою гілкою, робити код-рев'ю — для цього не потрібно залишати IDE. Можна настроїти інтеграцію з JetBrains Space, щоб переглядати та клонувати його репозиторії та виконувати код-рев'ю;
- Можливість ділитися налаштуваннями проекту - щоб дотримуватися єдиного стилю коду, можна поділитися з командою налаштуваннями стилю коду, використовуючи конфігурацію IDE, Prettier або EditorConfig. Також можна поділитися з колегами і іншими налаштуваннями проекту, наприклад, конфігураціями запуску.
- Інтеграція з баг-трекерами – передбачена можливість підключення WebStorm до баг-трекера та роботи над завданнями прямо з IDE.

Доступна можливість налаштовувати інтерфейс WebStorm під свої потреби, за допомогою різних тем і плагінів. Можливо також зберегти налаштування та використовувати їх в інших інсталяціях WebStorm. [12]

Для користувача доступні:

- Налаштування інтерфейсу - у WebStorm є кілька готових тем інтерфейсу, але можливо також виставити налаштування самостійно. Можна налаштувати видимість різних елементів інтерфейсу та поміняти їх розташування.
- Великий вибір розкладок - IDE пропонує поєднання клавіш майже для будь-якої дії. Можна використовувати будь-яку з існуючих розкладок чи створити власну. Також є можливість встановити розкладки інших редакторів, включаючи Vim, VS Code та Sublime Text .
- Плагіни – доступна можливість розширювати базову функціональність IDE і поповнювати набір налаштувань, використовуючи колекцію плагінів (див. рис. 2.6).

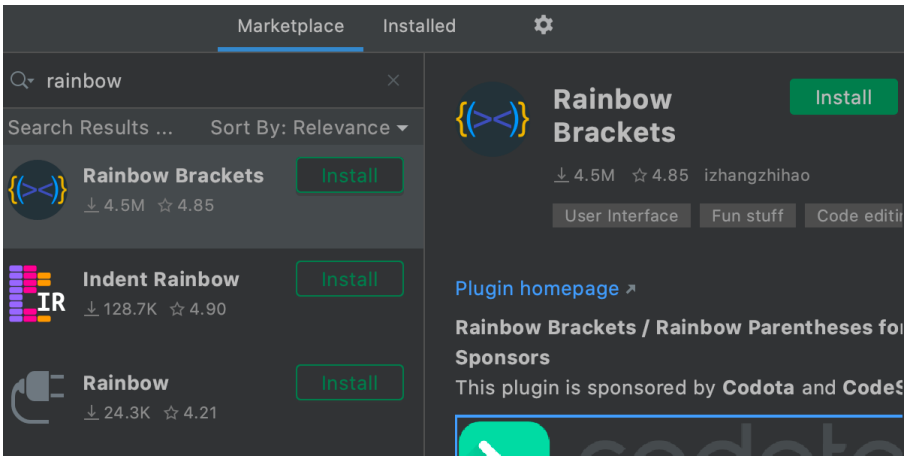

Рисунок – 2.6 - Колекція плагінів IDE

- Підтримка інструментів для роботи з базами даних та SQL (див. рис.2.7)

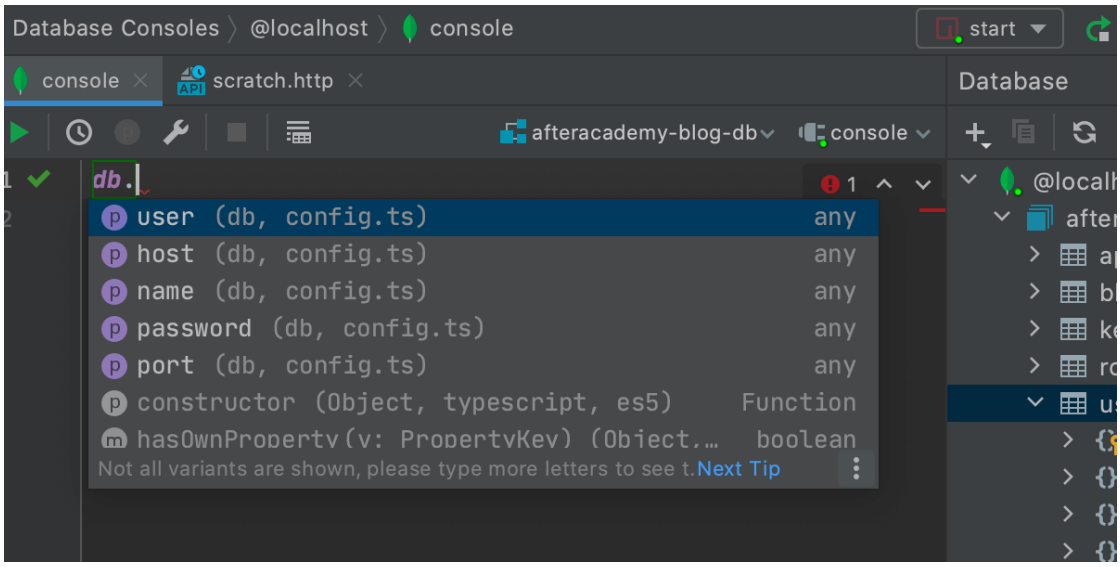

Рисунок 2.7 – Консоль для роботи з базами даних у IDE

- Спеціальні можливості - редактор пропонує безліч спеціальних можливостей для різноманітних потреб. Можна налаштовувати кольори елементів інтерфейсу, розмір вікон та шрифтів у редакторі, коригувати колірну схему у разі порушеного кольоросприйняття, поєднання клавіш та багато іншого.

- Режим швидкого редагування файлів - режим LightEdit дозволяє відкривати файл у полегшеному текстовому редакторі, що підтримує підсвічування синтаксису, автоматичне збереження та інші базові можливості.

Підводячи підсумок, можна сказати, що WebStorm – потужний редактор основних мов, який добре розпізнає структуру проекту, сканує ймовірні складності ще до відкриття розробки в браузері, рекомендуючи їх ефективне рішення. Продуктивності редактору надають включені IDE-інструменти для створення та тестування проектів. Редактор містить у собі масу зручних функцій для розробників. Виходячи з всього переліченого вище, було прийнято рішення використовувати для реалізації практичної частини проекту IDE WebStorm.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Інформаційна модель

Для реалізації поставленої задачі першим кроком було розроблено схеми функціонування веб-платформи для користувачів з ролями студент та ментор. Такі схеми мають наступну структури:

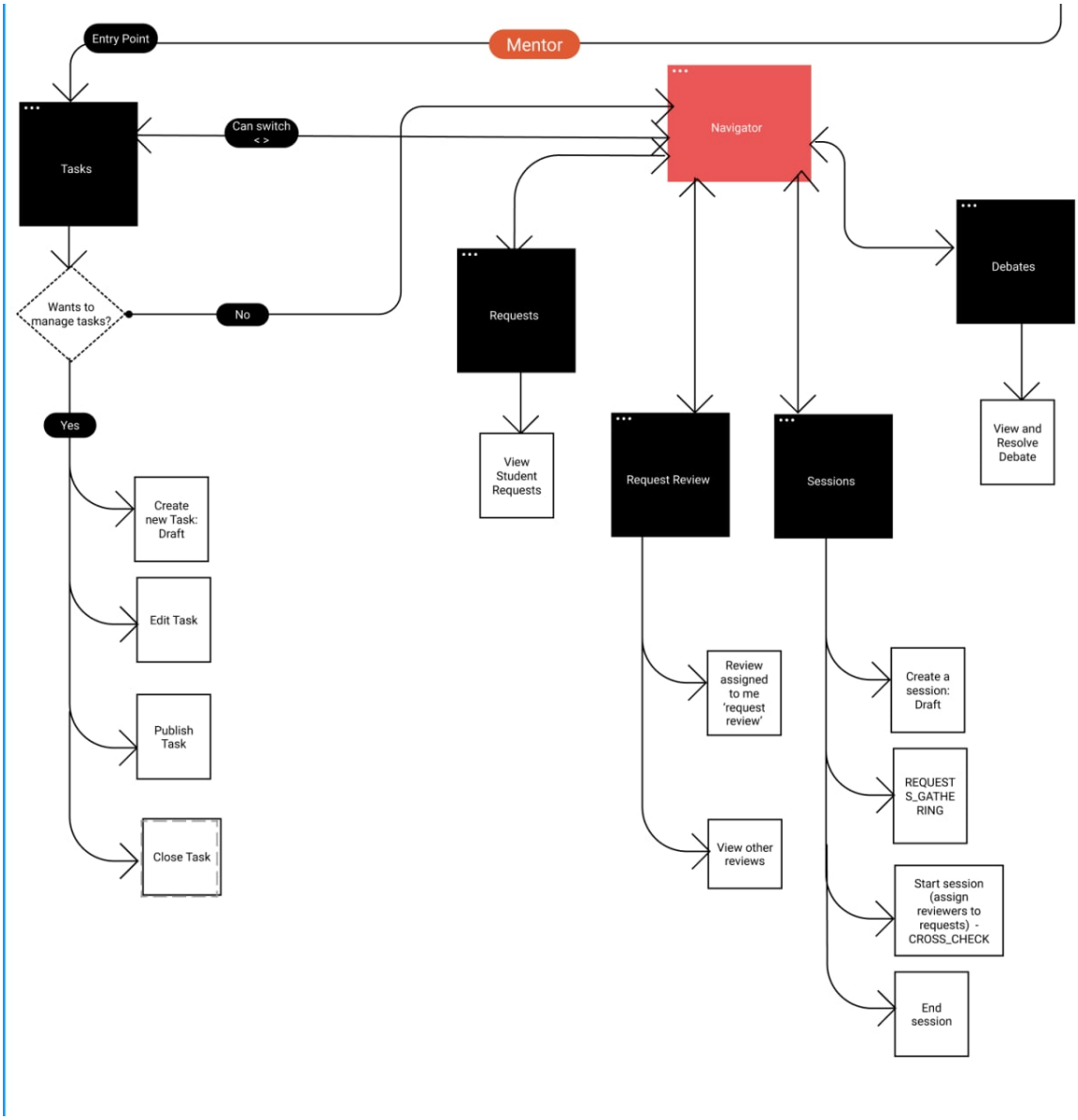

Рисунок 3.1 – Схема роботи додатку для користувача з роллю ментора

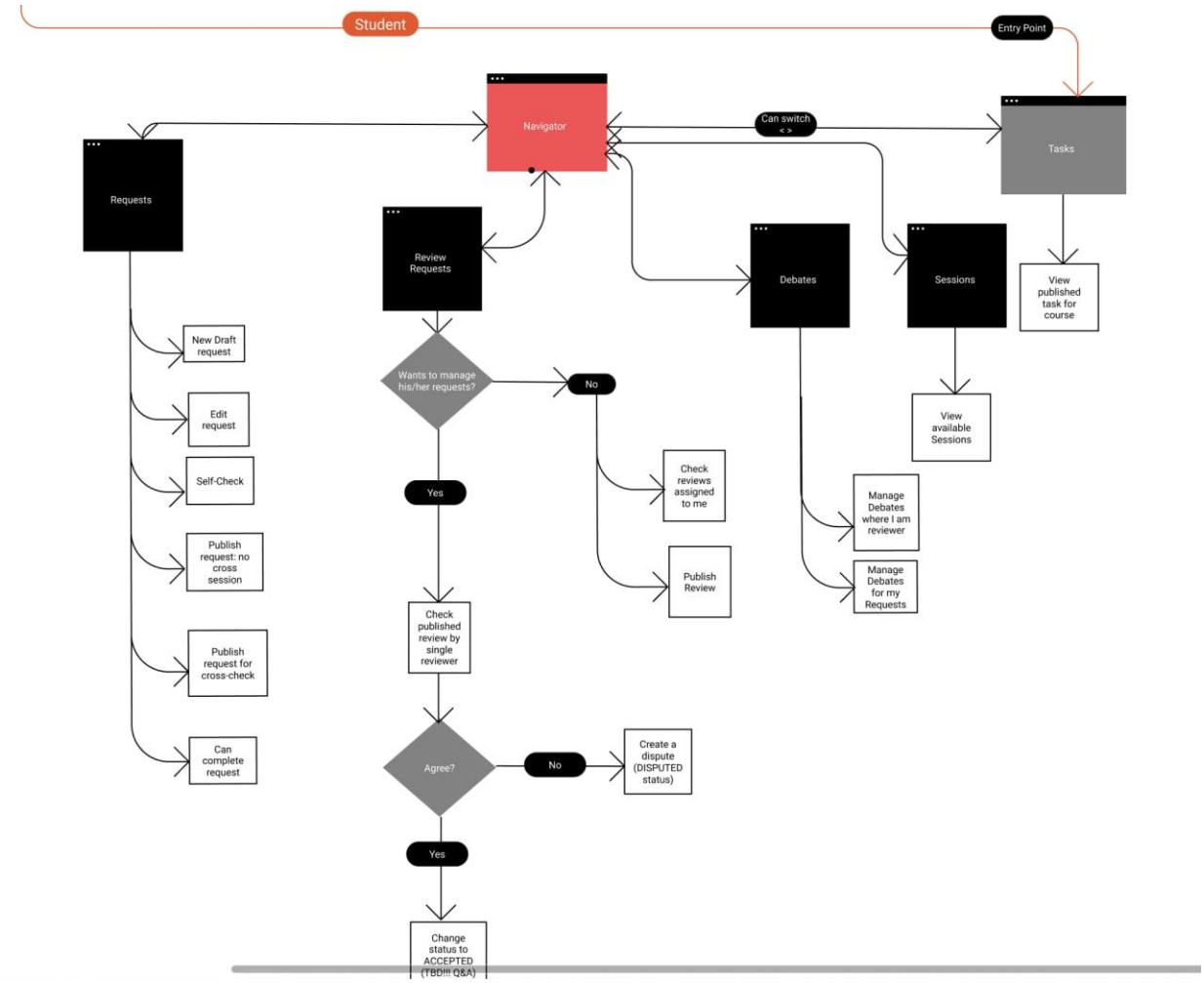

Рисунок 3.2 – Схема роботи додатку для користувача з роллю студента

Також дуже важливо розуміти, що функціональності студента та ментора можуть перетинатися та бути взаємозалежними. Далі було спроектовано структуру функціонування веб-сайту на базі вже створених сїем роботи додатку для ролей студент та ментор (див.рис.3.1 та рис.3.2). Створено можливості сортувати, створювати, редагувати, видаляти та відображати у табличному вигляді дан сутностей, які зберігаються на бекенді. Наприклад, задачі, запити на первірку, вже готові перевірки, сессії тощо. Загалом ці та інші можливості були закладені у сайт, на моменті його проектування.

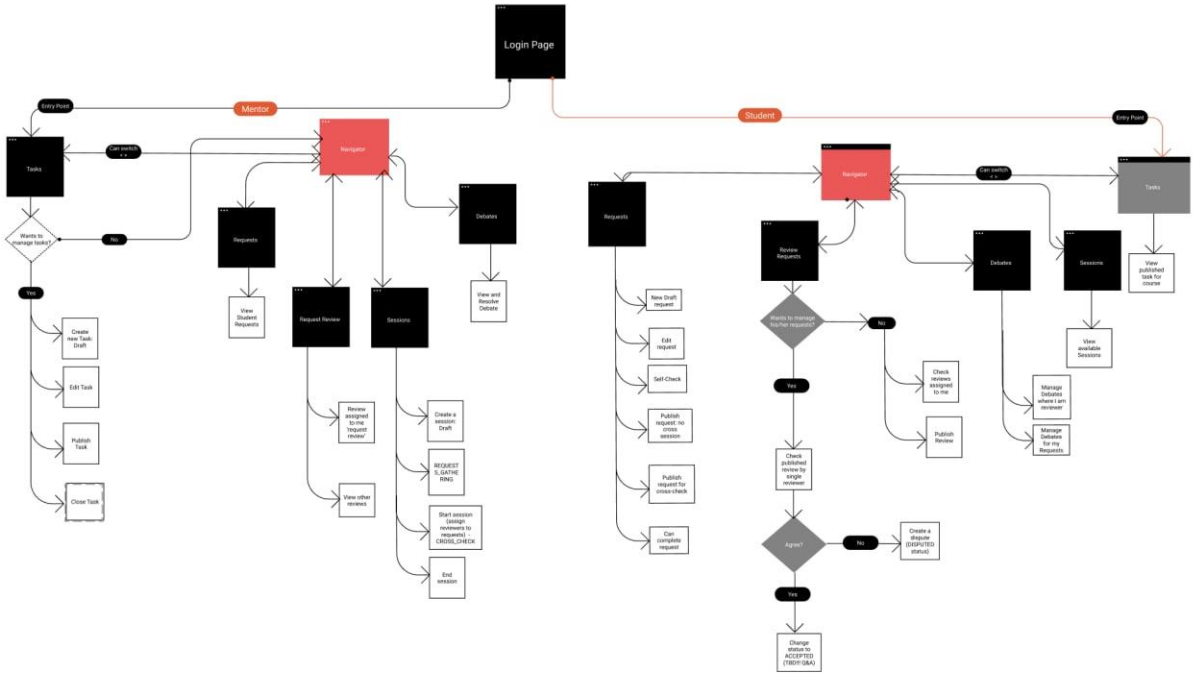

Рисунок 3.3 – Загальна схема функціонування додатку для користувачів

Наступним кроком було розроблено сутності даних, які будуть використані для забазпечення функціонування користувацького інтерфейсу. В якості технології баз даних була використана нереляційна база даних від Firebase, а сам сервіс Firebase був використаний як backend-частина функціонування додатку.

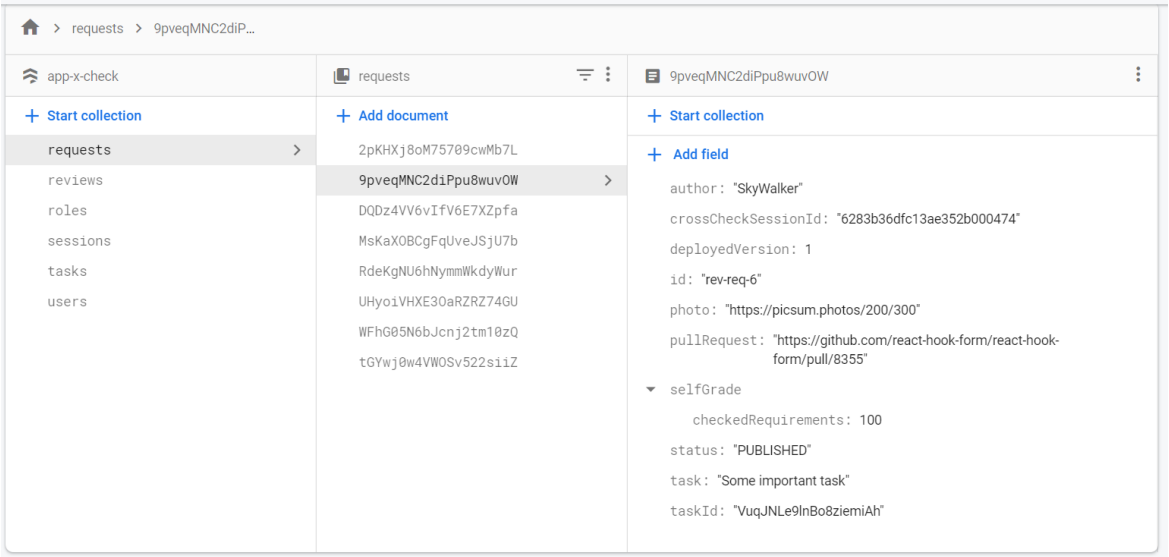

Рисунок 3.4 – Request entity

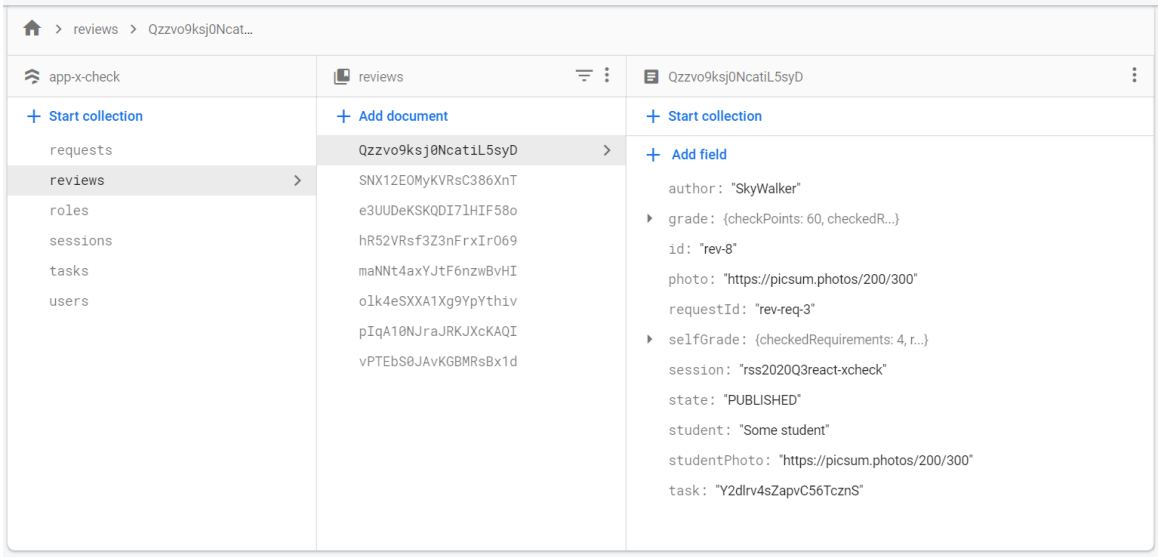

Рисунок 3.5 – Review entity

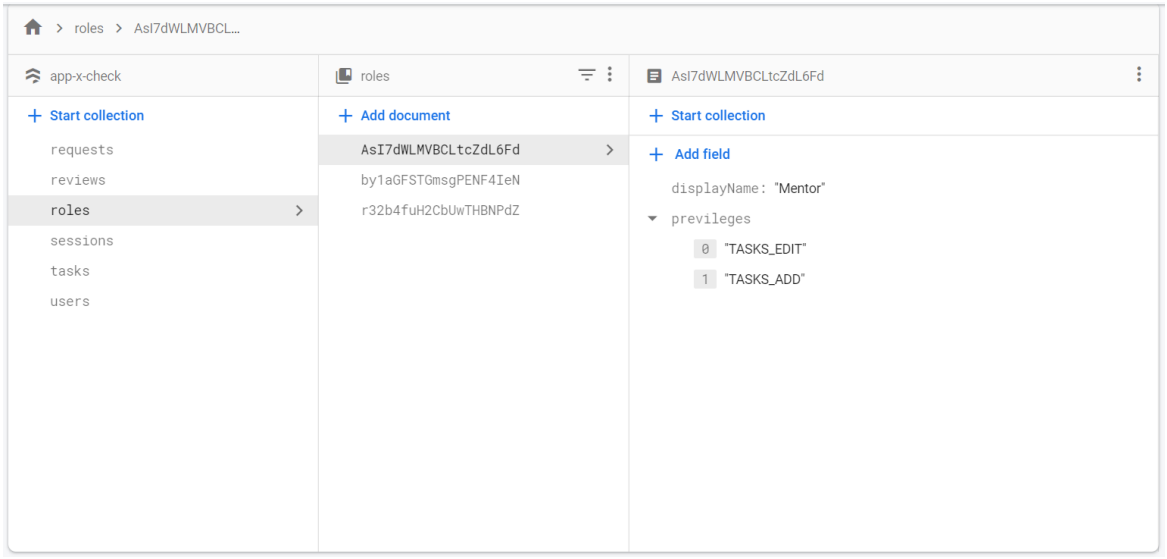

Рисунок 3.6 – Role entity

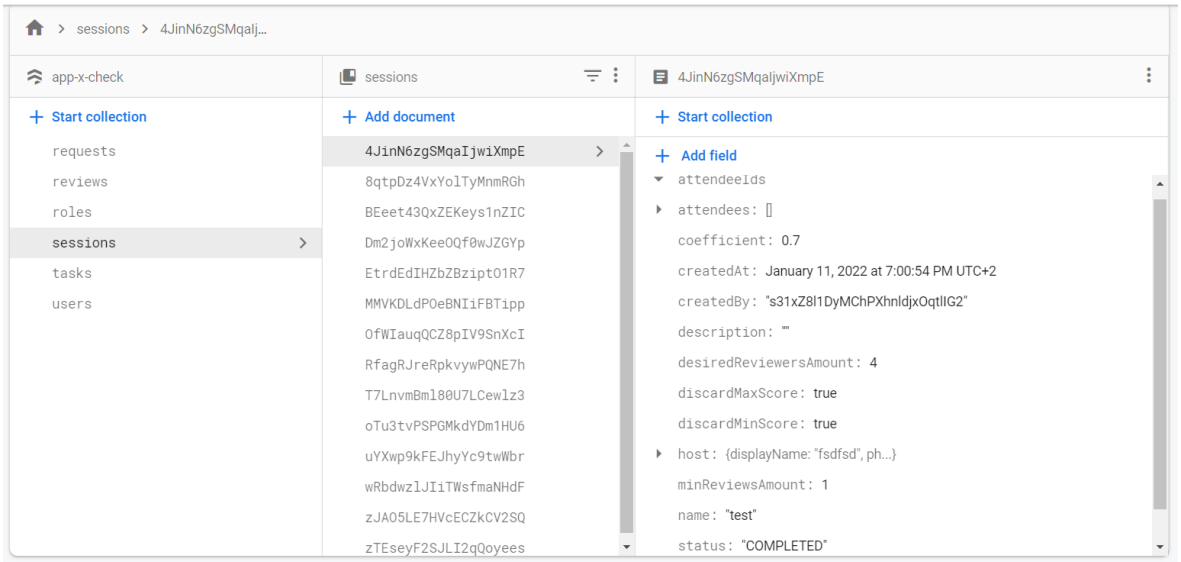

Рисунок 3.7 – Session entity

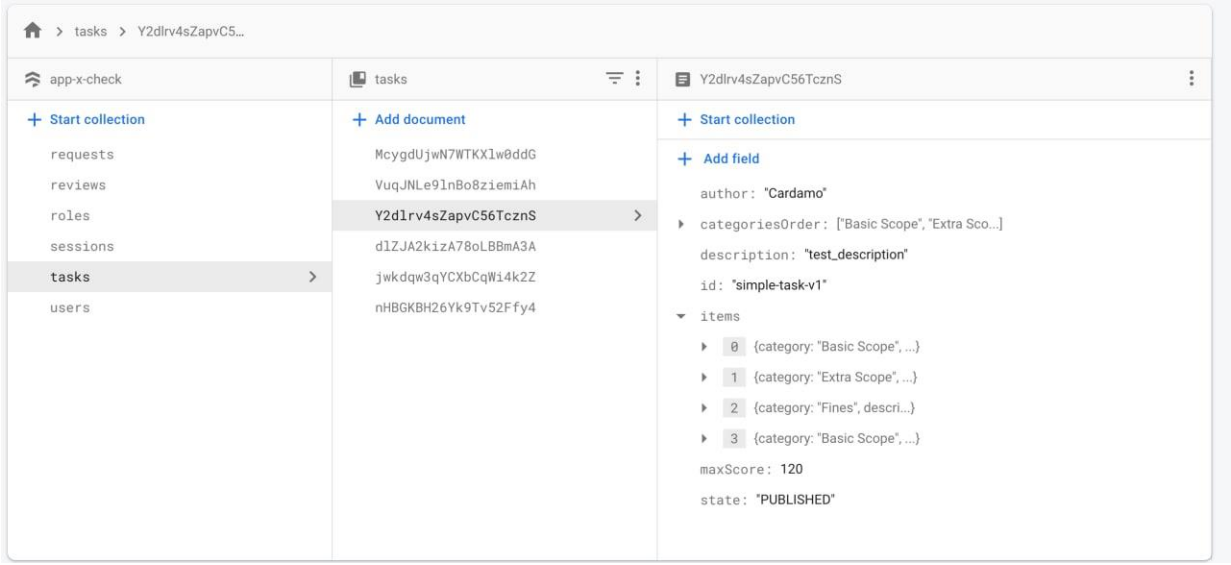

Рисунок 3.8 – Task entity

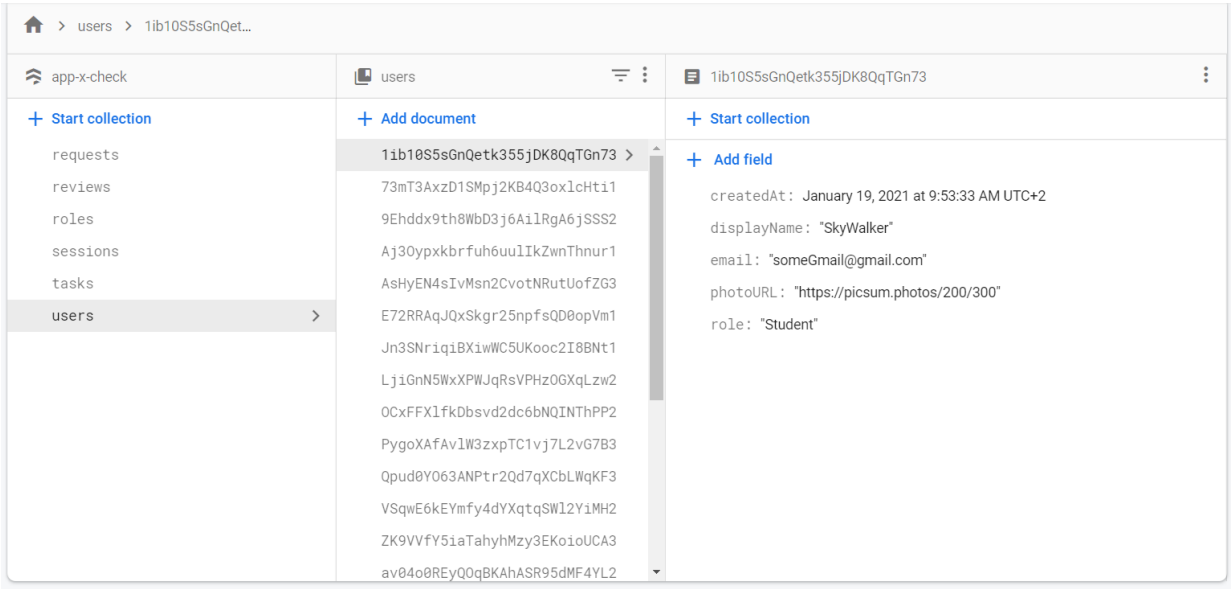

Рисунок 3.9 – User entity

3.2 Програмна релізація системи підвищенного захисту

Основною задачею системи підвищенного захисту у розробленому додатку – це обмеження доступу користувачів до певного контенту або до певного функціоналу на основі певних умов. Для того, щоб було систематичніше та більш організовано керувати такими умовми, також була розроблена система ролей для додатку. Усі вони використовують випробувану методологію під назвою Отже, контроль доступу на основі ролей має під собою 3 кроки реалізації.

Перший крок – це створення системи захисту маршрутизації у додатку.

// Створюємо конфігураційний файл, де ініціалізуємо сутніть роутингової системи

```
module.exports = {
    homePage: {
        route: '/',
        roles: ['student, 'mentor']
    },
    tasksPage: {
        route: '/tasks/',
        roles: ['student, 'mentor']
    },
    sessionsPage: {
        route: '/sessions,
        roles: ['student, 'mentor']
    },
    requestsPage: {
        route: '/requests',
        roles: ['student, 'mentor']
    },
    reviewsPage: {
        route: '/reviews,
        roles: ['student, 'mentor']
    }
};
```

// Створюємо компонент обгортку для роутів, в якому буде інкапсульовано логіку прорахунку ролі та подальшого рендеру або ж навпаки редіректу

```
const RouteWrapper = ({Component, ...props }) => {
   const auth = checkAuth();
// Отримання даних ролі
   let role = getUserRole(),
      access = checkAccess(role, props.roles);
// Перевірка на авторизованість
   if (!auth) {
      logout();
   } else if (auth && !access) {
      return <Redirect to="/error/401" />;
   }
// Повернення даних на основі ролі
   return (
      <Route
         {...props}
         render={routeProps => <Component {...routeProps} />}
      />
   );
};
// Створюємо головний компонент маршрутизатор у додатку
<Switch>
// Домашня сторніка
  <RouteWrapper
   exact
   path={routePaths.homePage.route}
   roles={routePaths.homePage.roles}
   Component={HomePage}
  />
```

```
// Сторінка задач
<RouteWrapper
    exact
    path={routePaths. tasksPage.route}
    roles={routePaths. tasksPage.roles}
    Component={TasksPage}
  />
// Сторініка сесій
  <RouteWrapper
    exact
    path={routePaths. sessionsPage.route}
    roles={routePaths. sessionsPage.roles}
    Component={SessionPage}
  />
// Сторінка запитів на перевірку
  <RouteWrapper
    exact
    path={routePaths. requestsPage.route}
    roles={routePaths. requestsPage.roles}
    Component={RequestsPage}
  />
// Сторінка перевірок
  <RouteWrapper
    exact
    path={routePaths. reviewsPage.route}
    roles={routePaths. reviewsPage.roles}
    Component={ReviewsPage}
  />
```

```
// Сторінки помилок
 <Route path="/error/:code">
   <ErrorComponent />
  </Route>
  <Route path="">
   <Redirect to="/error/404" />
  </Route>
</Switch>
```

Компонент обгортка роута перевірить дані аутентифікації та доступ на основі ролі, необхідної для маршруту та матриці ролей користувача. Якщо ви не пройшли автентифікацію, програма виведе вас із системи, щоб запобігти конфліктам. Якщо ви пройшли автентифікацію, але не маєте прав на доступ до маршруту, ми переспрямуємо вас на сторінку помилки. Якщо ви засвідчені та авторизовані, ми нарешті відобразимо компонент. Ця перевірка відбувається під час усіх змін маршрутів для наших захищених маршрутів.

Другий крок – це створення системи обмеженого доступу.

```
// Створюємо компонент обгортку-обмежувач
import { useSelector } from 'react-redux';
import { allowedUserChecker } from 'utils';
import PropTypes from 'prop-types';

const RBAC = ({ allowedRoles, children }) => {
   let userRoles = useSelector(state => state.userInfo.roles);
   let access = allowedUserChecker(userRoles, allowedRoles);
   return access && children;
};

RBAC.propTypes = {
```

```
  allowedRoles: PropTypes.arrayOf(PropTypes.string),
  children: PropTypes.element
};

export default RBAC;

// Імплементуємо компонент-обмежувач у додаток
import RBAC from './dir/RBAC';
...
<RBAC allowedRoles={[ 'ROLE_AUTHOR', 'ROLE_ADMIN']}>
  <Button type="submit" label="VERIFY AND PUBLISH" />
</RBAC>
```

Третій крок – це захист бекенд ендпоїнтів:

```
//
module.exports = (rolesWithAccess) => {
 let {roles} = req.user;
 // eg:  roles = ['author'] comming from JWT token
 return (req, res, next) => {
  if (rolesWithAccess.length && !rolesWithAccess.includes(roles)) {
   return res.status(401).json({ message: 'Unauthorized' });
   // send however you want this one way of sending it.
  } else {
   next();
  }
 }
};
```

3.3 Програмна реалізація функціонування навчальної веб-платформи

Кожна сторінка побудована за принципом логікаміститься в окремому та відображення також в окремому компоненті. Логіка взаємодії з бекендом побудована за допомогою створення запиту до бекенду на етапі монтування компоненту.

```
        // Взаэмодія з бекендом
        useEffect(() => {
  const db = firebase.firestore();
  let requests: any = [];

  db.collection('requests')
    .get()
    .then((query) => {
      query.forEach((doc) => {
        requests = [...requests, transformRequests(doc.data())];
      });
      setRequests(requests);
    });
}, []);
        // Організація логіки сторінки
        export const TaskDescription: React.FC = () => {
// Логіка обробки панелі
  const [panel, setPanel] = useState<string | string[]>('')
  const [panelItems, setPanelItems] = useState<Array<Iitem>>()
// Логіка обробки видимості задач
  const dispatch = useDispatch();
  const visible = useSelector((store: any) => store.taskStore?.isVisible)
  const id = useSelector((store: any) => store.taskStore?.id) || null
  const allTask = useSelector((taskStore: ItaskStore) =>
```

```
taskStore.firestore.data.tasks);
  const { Panel } = Collapse;

// Логіка обробки даних
  useEffect(() => {
    if(id){
    const task = allTask[id];
    const items = task?.items?.filter((item: Iitem) =>
    item.category === allTask[id].categoriesOrder[Number(panel)])
    setPanelItems(items)}
  }, [allTask, id, panel]);

 const onClose = () => {
    dispatch(deleteTask());
 };

// Рендер компоненту
 return (
  <Drawer placement='right' width={'640px'} closable={true}
onClose={onClose} visible={visible || false}>
   {id ?
      <div className="discription">
        <h3>Task Discription</h3>
        {visible ? allTask[id].description : null}
        <h3>Status</h3>
        {visible ? allTask[id].state : null}
        <h3>Scoring criteria</h3>
        <Collapse accordion onChange={(key) => setPanel(key)}>
          {visible ? allTask[id].categoriesOrder.map((el: string, idx: number) => (
            <Panel
```

```
            header={el}
            key={idx}
          >
           {panelItems ? panelItems.map((el,idx) => (
             <TaskDescriptionItem {...el} key={idx + 888} />
           )) : null }
          </Panel>
       )) : <Empty image={Empty.PRESENTED_IMAGE_SIMPLE} />}
        </Collapse>
      </div>

      :// Обробка відсутнхіх картинок
      <Empty image={Empty.PRESENTED_IMAGE_SIMPLE} />
      }
   </Drawer>
  );
};
```

ВИСНОВКИ

Після проведення аналізу предметної області та аналізу вимог до безпеки контенту в навчальних платформаї було прийнято рішення розробити власну програму навчальну платформу.

Як результат розробки – реалізовані методи та задіяні технологіх, за допомогою яких було досягнуто цілі створення навчальної веб-платформи, яка забезпечує менторів та студентів механізмами навчального процесу.

Створено схеми функціонування додатку для різних ролей та спроектовані сутності для роботи додатку. Розроблена система контролю доступу, яка базується на ролях користувачів та забезпечує безпеку контенту, функціоналу шляхом обмеження доступу за механізмом сконфігурованим в системі ролей додатку.

Під час розробки системи навчальної платформи були використанні різноманітні технології та знання, зокрема

- мова JavaScript;
- фреймворк React;
- сервіс Firebase;
- бібліотека Redux;
- команди Unix-подібних систем;
- веб-технології: HTML-розмітка, CSS-стилізація;
- підтримка програмного забезпечення: використання системи контролю версій, системи автоматичної збірки, уникнення антипатернів.

Розроблений додаток задовольняє всім вимогам, поставленим на етапі постановки завдання.

СПИСОК ЛІТЕРАТУРИ

1. Роберт Мартін. Чиста архітектура. Мистецтво розробки програмного забезпечення. 2018.

2. Дейв Крейн, Бер Бибо, Джордон Сонневельд. Ajax на практике. — М.: Вильямс, 2007. — ISBN 978-5-8459-1327-2.

3. Дэниел Вулстон. Ajax и платформа .NET 2.0 для профессионалов. — М.: Вильямс, 2007. — С. 464. — ISBN 1-59059-670-6.

4. Дейв Крейн, Эрик Паскарелло, Даррен Джеймс. AJAX в действии: технология — Asynchronous JavaScript and XML. — М.: Вильямс, 2006. — С. 640. — ISBN 1-932394-61-3.

5. Храмцов П.Б., Брик С.А., Русак А.М., Сурин А.И. Основы web-технологий Интернет-университет информационных технологий - ИНТУИТ.ру, 2003

6. Будилов В.А. Практические занятия по HTML. Краткий курс Наука и техника, 2001

7. Крамер Э. HTML: наглядный курс web-дизайна Диалектика, 2001

8. JavaScript: The Definitive Guide S.Spainhour, R.Eckstein Webmaster in a Nutshell. 2nd Edition O'Reilly, 1999

9. Azat Mardan. React Quickly: Painless web apps with React, JSX, Redux, and GraphQL. Manning, 2017

10. Mark Tielens Thomas. React in Action. Manning, 2018

11. Роббінс Д. HTML5, CSS3 та Javascript. Вичерпне керівництво, 2014

12. Stefan Rosca. WebStorm Essentials. Packt Publishing, 2015

13. React. A JavaScript library for building user interfaces [Електронний ресурс] – Режим доступу: https://reactjs.org/

14. Hoffman Andrew. Web Application Security: Exploitation and Countermeasures for Modern Web Applications, 2021

ДОДАТКИ

Додаток А. Лістинг програмного коду

```
// Tasks

import React, { useEffect, useState } from 'react';
import './Tasks.scss';
import { SearchOutlined } from '@ant-design/icons';
import { Table, Tag, Input, Button, Space } from 'antd';
import Highlighter from 'react-highlight-words';
import firebase from 'firebase';
import { TasksHeader } from './TasksHeader';
import { TaskDrawerContextState } from './TaskDrawer/TaskDrawerContext';
import { TaskDrawer } from './TaskDrawer/TaskDrawer';
import { TaskLayout } from './TaskCreate';
import { useFirestoreConnect } from 'react-redux-firebase';
import { useDispatch, useSelector } from 'react-redux';
import { ItaskStore } from 'interfaces/TaskInterface';
import { taskStatus } from 'enum/task.enums';
import { SessionsState } from 'interfaces/sessions-state.interface';
import { deleteTask, setTask, taskDescriptionVisible } from
'./TaskCreate/taskReducer/taskReducer';
import { ToastContainer } from 'react-toastify';
import { TaskDescription } from
'./TaskDrawer/TaskDescription/TaskDescription';

interface Tasks {
 key: string | number;
 taskName: string;
 status: string;
 updateTime: string;
 author: string;
 maxScore: number;
}

const transformTasks = (tasks: any, docId: string) => {
  const { id, state, lastUpdate, author, maxScore } = tasks;

 return {
```

```
  key: docId,
  taskName: id,
  status: state,
  updateTime: lastUpdate,
  author,
  maxScore
 };
};


export const Tasks = () => {
  const dispatch = useDispatch();
  const isLoadingData: boolean = useSelector((state: SessionsState) =>
state.firestore.status.requesting.tasks);

  const [ tasks, setTasks ] = useState<Array<Tasks> | Array<object>>([ {key:
'797984684'} ]);
  const [ selectedRowKeys, setSelectedRowKeys ] = useState<(string | number)[]
| undefined>([]);


  useFirestoreConnect([ { collection: 'tasks' } ]);

 const allTask = useSelector((taskStore: ItaskStore) =>
taskStore.firestore.data.tasks);

  useEffect(
    () => {
    selectedRowKeys?.length ? dispatch(setTask(allTask[selectedRowKeys[0]])) :
dispatch(deleteTask())
  },
  [allTask, dispatch, selectedRowKeys]
  );

 useEffect(() => {
  const db = firebase.firestore();

  let tasks: any = [];
```

```
 db.collection('tasks').get().then((query) => {
  query.forEach((doc) => {
   tasks = [ ...tasks, transformTasks(doc.data(), doc.id) ];
  });
  setTasks(tasks);
 });
}, [allTask]);


const onSelectChange = (selectedRowKeys: (string | number)[] | undefined) => {
   setSelectedRowKeys(selectedRowKeys);
};

const rowSelection = {
 selectedRowKeys,
 onChange: onSelectChange
};

const [ search, setSearch ] = useState({
 searchText: '',
 searchedColumn: ''
});
const { searchText, searchedColumn } = search;

const handleSearch = (selectedKeys: string, confirm: any, dataIndex: string) => {
 confirm();
 setSearch({
  searchText: selectedKeys[0],
  searchedColumn: dataIndex
 });
};

const handleReset = (clearFilters: any) => {
 clearFilters();
 setSearch((prevState: { searchText: string; searchedColumn: string }) => {
  return {
   ...prevState,
   searchText: ''
  };
```

```
  });
 };

 let searchInput: any;

 const getColumnSearchProps = (dataIndex: any) => ({
  filterDropdown: ({ setSelectedKeys, selectedKeys, confirm, clearFilters }: any)
=> (
   <div style={{ padding: 8 }}>
    <Input
     ref={(node) => {
      searchInput = node;
     }}
     placeholder={`Search task name`}
     value={selectedKeys[0]}
     onChange={(e) => setSelectedKeys(e.target.value ? [ e.target.value ] : [])}
     onPressEnter={() => handleSearch(selectedKeys, confirm, dataIndex)}
         style={{ width: 188, marginBottom: 8, display: 'block' }}
         key={`SearchInput${dataIndex}`}
    />
    <Space>
     <Button
      type='primary'
      onClick={() => handleSearch(selectedKeys, confirm, dataIndex)}
      icon={<SearchOutlined />}
      size='small'
           style={{ width: 90 }}
           key='SearchInputButtonSearch'
     >
      Search
     </Button>
     <Button key='SearchInputButtonReset' onClick={() =>
handleReset(clearFilters)} size='small' style={{ width: 90 }}>
      Reset
     </Button>
    </Space>
   </div>
  ),
  filterIcon: (filtered: any) => <SearchOutlined style={{ color: filtered ? '#1890ff'
```

```
: undefined }} />,
  onFilter: (value: any, record: any) =>
   record[dataIndex] ?
record[dataIndex].toString().toLowerCase().includes(value.toLowerCase()) : '',
  onFilterDropdownVisibleChange: (visible: any) => {
   if (visible) {
    setTimeout(() => searchInput.select(), 100);
   }
  },
  render: (text: any) =>
   searchedColumn === dataIndex ? (
    <Highlighter
     highlightStyle={{ backgroundColor: '#ffc069', padding: 0 }}
     searchWords={[ searchText ]}
     autoEscape
     textToHighlight={text ? text.toString() : ''}
    />
   ) : (
    text
   )
 });


 const renderStatus = (status: string) => {
    let color;

    switch (status) {
      case taskStatus.PUBLISHED:
        color = 'green';
        break;

      case taskStatus.PUBLISHED.toUpperCase():
        color = 'green';
        break;

      case taskStatus.DRAFT:
        color = 'orange';
        break;
```

```
      case taskStatus.DRAFT.toUpperCase():
        color = 'orange';
        break;
      default:
        color = 'default';
        break;
}

 return <Tag color={color}>{status?.toUpperCase()}</Tag>;
};

const filtersStatus = [
 {
  text: 'Published',
     value: 'Published',
     key: 'Published'
 },
 {
  text: 'Draft',
     value: 'Draft',
      key: 'Draft'
 },
 {
  text: 'Closed',
     value: 'Closed',
     key: 'Closed'
 }
];

 interface rows {
    [key: string]: string
 }

const columns = [
 {
  title: 'Task Name',
  dataIndex: 'taskName',
  key: 'taskName',
  ...getColumnSearchProps('taskName')
```

```
  },
  {
   title: 'Status',
   dataIndex: 'status',
   key: 'status',
   render: (status: string) => renderStatus(status),
   filters: filtersStatus,
   onFilter: (value: any, record: any) => record.status.indexOf(value) === 0
  },
  {
   title: 'Last Update',
   dataIndex: 'updateTime',
   key: 'updateTime'
  },
  {
   title: 'Author',
   dataIndex: 'author',
   key: 'author'
  },
  {
   title: 'Max Score',
   dataIndex: 'maxScore',
   key: 'maxScore'
    },
    {
    key: 'action',
    render: (values: rows) => (
      <Space size="middle">
        <Button type='default' onClick={() => dispatch(taskDescriptionVisible(true,
values.key))} >Description</Button>
      </Space>
    ),
  },
 ];

 return (
  <TaskDrawerContextState selectedRowKeys={selectedRowKeys}>
      <ToastContainer
      position="top-center"
```

```
      autoClose={5000}
      hideProgressBar={false}
      newestOnTop={false}
      closeOnClick
      rtl={false}
      pauseOnFocusLoss
      draggable
      pauseOnHover
    />
   <TaskDrawer>
    <TaskLayout />
   </TaskDrawer>
     <TaskDescription />
   <div className='tasks'>
    <TasksHeader />
    <div className='tasks-table'>
         <Table
           loading={isLoadingData}
           dataSource={tasks}
           columns={columns}
           rowSelection={rowSelection}
         />
    </div>
   </div>
  </TaskDrawerContextState>
 );
};

// Sessions

import React, { ReactText } from 'react';
import { Avatar, Table, Tag } from 'antd';
import { ColumnsType } from 'antd/es/table';
import { useFirestoreConnect } from 'react-redux-firebase';
import { useDispatch, useSelector } from 'react-redux';
import styles from './Sessions.module.scss';
import { Session } from '../../../../interfaces/app-session.interface';
import { COLORS, FRIENDLY_STATUS, SessionStatus } from
'../../../../enum/session-status.enum';
import { FirestoreSessionData } from '../../../../interfaces/firestore-
```

```
session.interface';
import SessionToolbar from './SessionToolbar/SessionToolbar';
import { UserOutlined } from '@ant-design/icons/lib';
import { setRowSelection } from './SessionsReducer';
import { SessionHost } from '../../../../interfaces/session-host.interface';
import SessionForm from './SessionForm/SessionForm';
import { SessionsRecord, SessionsState } from '../../../../interfaces/sessions-
state.interface';

const columns: ColumnsType<Session> = [
  {
    key: 'sessionName',
    title: 'Session',
    dataIndex: 'sessionName',
    ellipsis: true,
    sorter: (a: Session, b: Session) => {
      const x = a.sessionName.toUpperCase();
      const y = b.sessionName.toUpperCase();
      return x < y ? -1 : x > y ? 1 : 0;
    }
  },
  {
    key: 'taskName',
    title: 'Task',
    dataIndex: 'taskName',
    ellipsis: true,
    sorter: (a: Session, b: Session) => {
      const x = a.taskName.toUpperCase();
      const y = b.taskName.toUpperCase();
      return x < y ? -1 : x > y ? 1 : 0;
    }
  },
  {
    key: 'status',
    title: 'Status',
    dataIndex: 'status',
    filters: Object.values(FRIENDLY_STATUS).map(el => ({ text: el, value: el
})),
    onFilter: (value: any, record: Session) => record.status.indexOf(value) === 0,
```

```
    render: (tag: SessionStatus) => (
      <>{tag &&
      <Tag color={COLORS[tag]} key={tag}>
        {tag.toUpperCase()}
      </Tag>
      }
      </>
    )
  },
  {
    key: 'qty',
    title: '# of attendees',
    dataIndex: 'qty',
    sorter: (a: Session, b: Session) => a.qty - b.qty,
    render: (tag: number) => (
      <Tag color='default' key={tag}>
        {tag}
      </Tag>
    )
  },
  {
    key: 'user',
    title: 'Lead',
    dataIndex: 'user',
    sorter: (a: Session, b: Session) => {
      const x = a.user?.displayName || 'Anonymous';
      const y = b.user?.displayName || 'Anonymous';
      return x < y ? -1 : x > y ? 1 : 0;
    },
    render: (user: SessionHost) => (
      <div className={styles.user}>
        {user?.photoURL ? <Avatar src={user.photoURL}
classNameName={styles.user__avatar}/> :
          <Avatar icon={<UserOutlined/>} classNameName={styles.user__avatar}/>}
        {user?.displayName ? <span>{user?.displayName}</span> :
<span>Anonymous</span>}
      </div>
    )
  }
```

```
];

export default function Sessions() {
  const dispatch = useDispatch();
  const sessions: SessionsRecord = useSelector((state: SessionsState) =>
state.firestore.data.sessions);
  const isLoadingData: boolean = useSelector((state: SessionsState) =>
state.firestore.status.requesting.sessions);

  useFirestoreConnect([
    {
      collection: 'tasks',
      where: [
        ['state', '==', 'PUBLISHED']
      ],
      storeAs: 'publishedTasks'
    }, {
      collection: 'sessions'
    }
  ]);

  function getModifiedSessionData(): Session[] {
    const modifiedData: Session[] = [];
    if (sessions) {
      Object.keys(sessions).forEach((el: string) => {
        if (sessions[el]) {
          const values: FirestoreSessionData = sessions[el];
          modifiedData.push({
            key: el,
            sessionName: values?.name,
            taskName: values?.task?.taskName,
            qty: values?.attendees?.length || 0,
            status: FRIENDLY_STATUS[values?.status as SessionStatus],
            user: values?.host,
            task: values.task
          });
        }
      });
    }
```

```
    return modifiedData;
  }

  return (
    <div>
      <div className={styles.toolbar}>
        <SessionToolbar/>
      </div>
      <div className={styles.main}>
        <Table loading={isLoadingData} columns={columns} style={{ width:
'100%' }}
               dataSource={getModifiedSessionData()}
               showSorterTooltip={false}
               pagination={{ pageSize: 10 }}
               rowSelection={{
                 onChange: (selectedRowKeys: ReactText[]) => {
                   dispatch(setRowSelection(selectedRowKeys));
                 }
               }}
        />
      </div>
      <SessionForm/>
    </div>
  );
}
```

// Reviews

```
import React, { useEffect, useState } from 'react';
import styles from '../Sessions/Sessions.module.scss';
import { Table } from 'antd';
import { AppReviewInterface } from '../../../../interfaces/app-review.interface';
import firebase from 'firebase';
import { columnsRequests } from './reviewTableDefinition';
import ReviewsToolBar from './ReviewsToolBar/ReviewsToolBar';
import { ReviewStatusEnum } from '../../../../enum/review-status.enum';



const Reviews = () => {
```

```
const [reviews, setReviews] = useState<AppReviewInterface[]>([]);
const db = firebase.firestore();
let reviewCounter = 0;

useEffect(() => {
  const db = firebase.firestore();
  db.collection('reviews').get()
    .then((reviews) => {
      let rvs: any[] = [];
      reviews.forEach((r) => {
        rvs.push(Object.assign({}, r.data(), {key: r.data().id + r.data().requestId}));
      })
      setReviews(rvs);
    })

}, [reviewCounter])

const addRowHandler = () => {
  const newRow = {
    key: "0007",
  id: "0007",
  requestId: "ddddd",
  author: "Mr.Bean",
  state: ReviewStatusEnum.REJECTED
  }

  db.collection('reviews').add(newRow)
    .then(() => {
      reviewCounter++;
    });
}

return (
  <div>
    <h1>Reviews</h1>
    <ReviewsToolBar
      addRow={addRowHandler}
    />
```

```
      <div className={styles.main}>
        <Table columns={columnsRequests} style={{ width: '100%' }}
              dataSource={reviews}
              pagination={{ pageSize: 10 }}
        />
      </div>
    </div>
  );
};

export default Reviews;

// SelfCheck

import React from 'react';
import { Drawer, Form, Collapse } from 'antd';
import 'antd/dist/antd.css';
import './Selfcheck.scss';
import FormHeader from '../FormHeader/FormHeader';
import { useSelector } from 'react-redux';
import { useFirestoreConnect} from 'react-redux-firebase';
import CategoryItem, { TaskItem } from './CategoryItem';

const { Panel } = Collapse;

interface SelfcheckProps {
  isVisible: boolean,
  hide: () => void,
  setselfGradeValues: (values: any) => void,
  form: any,
  taskId: string,
  totalPoints: number,
  checkedRequirements: number,
  setTotalPoints: (number: number) => void,
  setCheckedRequirements: (number: number) => void,
}

interface TasksState {
  firestore: {
    data: {
```

```
    tasks: any,
   }
  }
}


const Selfcheck = (props: SelfcheckProps) => {
  const { isVisible, hide, setselfGradeValues, form, taskId, totalPoints,
setTotalPoints, checkedRequirements, setCheckedRequirements } = props;
  useFirestoreConnect([{ collection: 'tasks' }]);
  const tasks = useSelector((state : TasksState) => state.firestore.data.tasks);

  const handleClose = () => {
   hide();
   form.resetFields();
   setTotalPoints(0);
   setCheckedRequirements(0);
  }

  const onFinish = (values: object) => {
   console.log('Received values of selfcheck form: ', values);
   hide();
   addSelfGrade(values);
  };

  const onValuesChange = (changedValues: object, allValues: any) : void => {
   console.log(allValues);
   setCurrentValues(allValues);
   form.setFieldsValue(allValues);
   console.log(form.getFieldValue());
  }

  const setCurrentValues = (values: any) => {
   let totalPoints = 0;
   let checkedRequirements = 0;
   Object.keys(values).forEach((key: string) => {
    if (typeof values[key] === 'number') {
     totalPoints += values[key];
     checkedRequirements++;
```

```
    }
   });
   setTotalPoints(totalPoints);
   setCheckedRequirements(checkedRequirements);
  }

  const addSelfGrade = (values: any) => {
   Object.keys(values).forEach((key: string) => {
    if (values[key] === undefined) {
     delete values[key];
    }
   });

   setselfGradeValues({
    ...values,
    totalPoints,
    checkedRequirements
   });
  }

  return (
   <Drawer
    mask={false}
    closable={false}
    visible={isVisible}
    placement='left'
    width={600}
    title={
     <FormHeader title="Self-check" onClose={handleClose} form={form}/>
   }
   >
   <div className="self-check">
     <Form name="self-check" form={form} onFinish={onFinish}
onValuesChange={onValuesChange} initialValues={undefined} >
      <div className="self-check__current-values">
        <h3>Total points: {totalPoints}/{isVisible &&
tasks[taskId].maxScore}</h3>
        <h3>Checked requirements: {checkedRequirements}/{isVisible &&
tasks[taskId].items.length}</h3>
```

```
        </div>
        <p>{(tasks && isVisible) && tasks[taskId].description}</p>
        <Collapse bordered={false} style={{backgroundColor: 'white'}}>
         {(tasks && isVisible) &&
            tasks[taskId].categoriesOrder.map((category: string) => (
              <Panel header={category} key={category}>
                {tasks[taskId].items.map((item: TaskItem, ind: number) => {
                  return item.category === category && <CategoryItem item={item}
key={item.id} />;
                }
                )}
              </Panel>
            ))
         }
        </Collapse>
       </Form>
    </div>
    </Drawer>
  );
}

export default Selfcheck;
```

// Requests

```
import React, { useState } from 'react';
import styles from './Requests.module.scss';
import './Requests';

import HeaderRequests from './HeaderRequests/HeaderRequests';
import TopPanelRequests from './TopPanelRequests/TopPanelRequests';
import TableRequests from './TableRequests/TableRequests';

import { Button, Form } from 'antd';
import { EditOutlined } from '@ant-design/icons';
import RequestForm from '../RequestForm/RequestForm';

export const Requests = () => {
  const [isVisible, setVisibility] = useState(false);
  const [form] = Form.useForm();
```

```
  const handleClose = () => {
        setVisibility(false);
        form.resetFields();
  }

  return (
    <div className={styles.Requests__container}>
      <HeaderRequests />

      <Button
        className={styles.Requests__btn}
        icon={<EditOutlined />}
        onClick={() => setVisibility(true)}>
        Show request form
      </Button>
      <RequestForm isVisible={isVisible} onClose={handleClose} form={form}/>
      <TopPanelRequests />
      <div>
        <TableRequests />
      </div>
    </div>
  );
};
```