МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХЕРСОНСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
КАФЕДРА ІНФОРМАТИКИ, ПРОГРАМНОЇ ІНЖЕНЕРІЇ ТА
ЕКОНОМІЧНОЇ КІБЕРНЕТИКИ

Проектування бекенд частини та розробка прототипу веб-додатку редагування розкладу

Курсова робота з програмування

Виконавець: студент 4 курсу 431 групи
напряму підготовки «Інформатика»
Сенчишен Денис Олександрович

Науковий керівник: кандидат педагогічних наук,
доцент Кушнір Наталія Олександрівна

Херсон — 2018

ЗМІСТ

ВСТУП

Якість підготовки спеціалістів у закладах освіти і особливо ефективність використання науково-педагогічного потенціалу залежать певною мірою від рівня організації навчального процесу.

Одна з основних складових цього процесу — розклад занять — регламентує трудовий ритм, впливає на творчу віддачу викладачів, тому його можна вважати фактором оптимізації використання обмежених ресурсів — викладацького складу і аудиторного фонду.

Проблему складання розкладу слід розглядати не тільки як трудомісткий процес, об'єкт автоматизації з використанням комп’ютера, але і як проблему оптимального керування. Оскільки всі фактори, що впливають на розклад, практично неможливо врахувати, а інтереси учасників навчального процесу різноманітні, задача складання розкладу є багатокритеріальною з нечіткою множиною факторів.

Незалежно від алгоритму побудови розкладу, виникає прикладна проблема з інструментів різних рівнів, що використовуються в процесі. Саме ним і буде присвячено проведену роботу.

Актуальність дослідження полягає в необхідності забезпечення всіх учасників освітнього процесу доступом до актуальної версії розкладу занять у будь-який час, а також можливості спрощення процесу формування розкладу та подальшої інформатизації освітнього процесу.

Об’єкт дослідження — системи для планування розкладу. Предмет дослідження — веб-додаток для планування розкладу в закладах освіти з поділом учнів (вихованців, здобувачів освіти тощо) на стабільні академічні групи.

Метою роботи є проектування розширюваного веб-додатку редагування розкладу в закладах освіти з можливістю використання всіма учасниками освітнього процесу та розробка його робочого прототипу.

Для реалізації мети поставлено наступні завдання:

1. Проаналізувати характеристики існуючих систем планування, зокрема обсяг їх можливостей.
2. Проаналізувати окремі частини процесу підготовки розкладу на прикладі факультету комп’ютерних наук, фізики та математики ХДУ.
3. На основі проведеного аналізу розробити вимоги щодо можливостей додатку та його інтерфейсу.
4. Відповідно до створених вимог розробити проект додатку та бекенд частини.
5. Розробити робочий прототип бекенд частини (зокрема реалізувати структуру бази даних та окремі частини API) і інтерфейсу додатку.
6. Розробити документацію до публічного API
7. Обґрунтувати використані технології при проектуванні бекенд і фронтенд частин.

Очікується, що спроектований продукт буде придатний до використання всіма учасниками освітнього процесу в ЗВО.

Робота складається з 4 розділів, містить 17 рисунків.

1. РОЗДІЛ 1. АНАЛІЗ СИСТЕМ ПЛАНУВАННЯ

1.1. Історія виникнення комп’ютерних систем планування

Ідея планування робіт існує стільки, скільки існує людська цивілізація, адже ще в неоліті, з переходом до тваринництва і землеробства, постають задачі з контролем циклічних процесів, що і викликало у подальшому створення календаря і писемності для фіксування задач.

1.2. Сучасні сервіси для планування завдань

На сьогодні, кожна людина так чи інакше стикається у повсякденному житті з системами, пов'язаними з контролем часу.

1.2.1. Microsoft Outlook

Microsoft Outlook — додаток-органайзер, входить в пакет офісних програм Microsoft Office. Дозволяє працювати з електронною поштою, надає функції календаря, планувальника завдань, записника і менеджера контактів. Крім того, Outlook дозволяє відстежувати роботу з документами пакету Microsoft Office для автоматичного складання щоденника роботи.

Outlook може використовуватися і як окремий додаток, так і виступати в ролі клієнта для Microsoft Exchange Server, що надає додаткові функції для спільної роботи всіх користувачів організації: загальні поштові скриньки, папки завдань, календарі, планування часу загальних зустрічей, узгодження документів тощо.

Крім цього, дозволяє підключати через протоколи POP3/IMAP інші поштові сервіси та додатки, що надаються ними. Зокрема, нижче буде розглянуто синхронізацію MS Outlook з сервісами Google.

1.2.2. Lightning

Lightning — проект Mozilla Foundation, що додає функції календаря і планувальника в Mozilla Thunderbird — безкоштовну кросплатформну програму для роботи з електронною поштою і новинами, що може вважатися відкритим аналогом для відповідних продуктів з пакету Microsoft Office.

1.2.3. Google Calendar

Google Calendar — безкоштовний веб-додаток для тайм-менеджменту розроблений Google. Інтерфейс подібний до аналогічних календарних додатків, таких як Microsoft Outlook. Має різні режими перегляду, зокрема денний, тижневий та місячний. Події зберігаються онлайн, а тому календар можна переглядати з будь-якого пристрою, обладнаного доступом до мережі Інтернет. Додаток може імпортувати та експортувати файли календаря різних форматів, а для існуючих — задавати різні права доступу.

Слід зазначити, що.Google Calendar, як і інші сервіси Google, має відкрите API, що дозволяє взаємодіяти з ним через власні додатки після відповідних налаштувань.

1.2.4. Корпоративні інформаційні системи

Корпоративна інформаційна система — це інформаційна система, яка підтримує автоматизацію функцій управління на підприємстві і постачає інформацію для прийняття управлінських рішень. У ній реалізована управлінська ідеологія, яка об'єднує бізнес-стратегію підприємства і прогресивні інформаційні технології [2].

У загальному визначенні «автоматизована система» — сукупність керованого об’єкта й автоматичних керувальних пристроїв, у якій частину функцій керування виконує людина. Вона представляє собою

організаційно-технічну систему, що забезпечує вироблення рішень на основі автоматизації інформаційних процесів у різних сферах діяльності. Сучасні автоматизовані системи управління навчальним процесом у вищих навчальних закладах здатні вирішувати велику кількість функцій, а саме [3]:

- планування, контроль та аналіз навчальної діяльності;
- оперативний доступ до інформації про навчальний процес;
- єдину систему звітів, як внутрішніх, так і за вимогами МОН України;
- системи безпеки даних з урахуванням вимог законодавства;
- облік контингенту студентів та співробітників;
- проведення вступної кампанії;
- формування пакетів даних з метою виготовлення тих чи інших документів.

Функціонування будь-якої автоматизованої системи можна швидко адаптувати до особливостей навчального процесу конкретного навчального закладу, до локальних мереж різного рівня, що допомагає розширити коло користувачів (адміністрації, викладачів і студентів) для оперативного забезпечення їх необхідною інформацією. Отже, використання таких систем дає змогу не тільки удосконалити якість планування навчального процесу, а й оперативність управління ним [3].

Не зважаючи на всі переваги, які надає використання автоматизованих систем, досі далеко не в кожному закладі вони впроваджені чи використовуються в повній мірі з тих чи інших причин — інерційності поглядів адміністрації, супротив працівників або «саботаж» на місцях, відсутність фінансової або організаційної можливості.

В ХДУ використовується корпоративна інтегрована система «Інформаційно-аналітична система (IAS)». Вона дозволяє вести облік

працівників і студентів, бухгалтерський облік, контроль за матеріальними цінностями.

Система дозволяє вносити і ефективно стежити за будь-якими змінами. В основі системи лежить ядро, на основі ядра виконується розширення системи до будь-якої кількості компонентів. При цьому основна функціональність може бути розширена за рахунок додаткових компонентів.

Програма IAS орієнтована на платформу Windows з використанням MS SQL Server. Вона має багаторівневу архітектуру, що складається з бази даних, бізнес-логіки та клієнтського інтерфейсу. Внутрішній журнал реєстрації подій дозволяє вести та слідкувати за записами, що стосуються усіх подій.

Відсутність компонентів, пов’язаних з формуванням розкладу занять, та відсутність у використанні сторонніх рішень ставить задачу з проектування власного додатку для забезпечення всіх учасників освітнього процесу доступом до актуальної версії розкладу занять у будь-який час, а також можливості спрощення процесу формування розкладу та подальшої інформатизації освітнього процесу

2. РОЗДІЛ 2. ПРОЕКТУВАННЯ БАЗИ ДАНИХ

2.1. Історія виникнення комп’ютерних баз даних

База даних – сукупність даних, організованих відповідно до певної прийнятої концепції, яка описує характеристику цих даних і взаємозв'язки між їхніми елементами. Дані у базі організовують відповідно до моделі організації даних.

В загальному випадку базою даних можна вважати будь-який впорядкований набір даних, наприклад, паперову картотеку бібліотеки. Але все частіше термін «база даних» використовуєтьсяу контексті використання баз даних в інформаційних системах, як і самі бази даних переносяться в електронні системи в процесі інформатизації. На даний час додатки для роботи з базами даних є одними з найпоширеніших прикладних програм [4].

Через тісний зв'язок баз даних з системами керування базами даних (СКБД) під терміном «база даних» нерідко неточно мається на увазі система керування базами даних. Але варто розрізняти базу даних — сховище даних, та СКБД — засоби для роботи з базою даних. Надалі, в роботі під терміном «база даних», в залежності від контексту, може матися на увазі як сукупність даних чи певні її параметри, так і СКБД, крім випадків де це не очевидно.

Розроблення перших баз даних розпочинається в 1960-ті роки. Переважно, дослідницькі роботи ведуться в проектах IBM та найбільших університетів. Пізніше, на початку 1970-х років Едгар Ф. Кодд обґрунтовує основи реляційної моделі [5]. Уперше цю модель було використано у бази даних Ingres та System R, що були лише дослідними прототипами. Проте вже в 1980-ті рр. з’являються перші комерційних версій реляційних БД Oracle та DB2. Реляційні бази даних починають успішно витісняти мережні

та ієрархічні. Починаються дослідження розподілених (децентралізованих) баз даних.

2.2. Реляційні бази даних

Реляційна модель даних — логічна модель даних, вперше описана Едгаром Ф. Коддом [5]. В даний час ця модель є фактичним стандартом, на який орієнтуються більшість сучасних СКБД.

У реляційній моделі досягається більш високий рівень абстракції даних, ніж в ієрархічній або мережевій. Стверджується, що «реляційна модель надає засоби опису даних на основі тільки їх природної структури, тобто без потреби введення якоїсь додаткової структури для цілей машинного представлення» [5]. А це означає, що подання даних не залежить від способу їх фізичної організації, що забезпечується за рахунок використання математичного поняття відношення.

До складу реляційної моделі даних зазвичай включається теорія нормалізації. Дейт визначив наступні частини реляційної моделі даних [6]:

- структурна;
- маніпуляційна;
- цілісна.

Структурна частина моделі визначає, що єдиною структурою даних є нормалізоване n-арне відношення.

2.2.1. Нормалізація бази даних

Нормалізація схеми бази даних — процес розбиття одного відношення (таблиці в поняттях СУБД) відповідно до алгоритму нормалізації на кілька відношень на основі функціональних залежностей.

Нормальна форма визначається як сукупність вимог, яким має задовольняти відношення, з точки зору надмірності, яка потенційно може призвести до логічно помилкових результатів вибірки.

Таким чином, схема реляційної бази даних покроково, у процесі виконання відповідного алгоритму, переходить у першу, другу, третю і так далі нормальні форми. Якщо відношення відповідає критеріям n-ої нормальної форми та всіх попередніх нормальних форм, тоді вважається, що це відношення знаходиться у нормальній формі n-ого рівня.

2.2.2. СКБД PostgreSQL

PostgreSQL — широко розповсюджена система керування базами даних з відкритим вихідним кодом. Прототип був розроблений в Каліфорнійському університеті Берклі в 1987 році, пізніше проект Берклі було зупинено, а реалізацію було викладено в Інтернет під назвою Postgres95 після вдосконалення вихідного коду. Наразі підтримкою й розробкою займається група спеціалістів, які добровільно приєднались до проекту.

Сервер PostgreSQL написаний на мові C. Розповсюджується у вигляді вихідного коду, який необхідно відкомпілювати. Разом з кодом розповсюджується детальна документація.

2.3. Технологія ORM

ORM — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних». В об'єктно-орієнтованому програмуванні об'єкти в програмі представляють об'єкти з реального світу.

Суть проблеми полягає в перетворенні таких об'єктів у форму, в якій вони можуть бути збережені у файлах або базах даних, і які легко можуть бути витягнуті в подальшому, зі збереженням властивостей об'єктів і відношень між ними. Ці об'єкти називають «постійними». Існує кілька підходів до розв'язання цієї задачі. Деякі пакети вирішують цю проблему, надаючи бібліотеки класів, здатних виконувати такі перетворення

автоматично. Маючи список таблиць в базі даних і об'єктів в програмі, вони автоматично перетворять запити з одного вигляду в інший.

В проекті використано ORM Sequelize. Спроектовано на реалізовано у вигляді моделей та відповідним їм таблиць структуру бази даних (рис. 2.1).

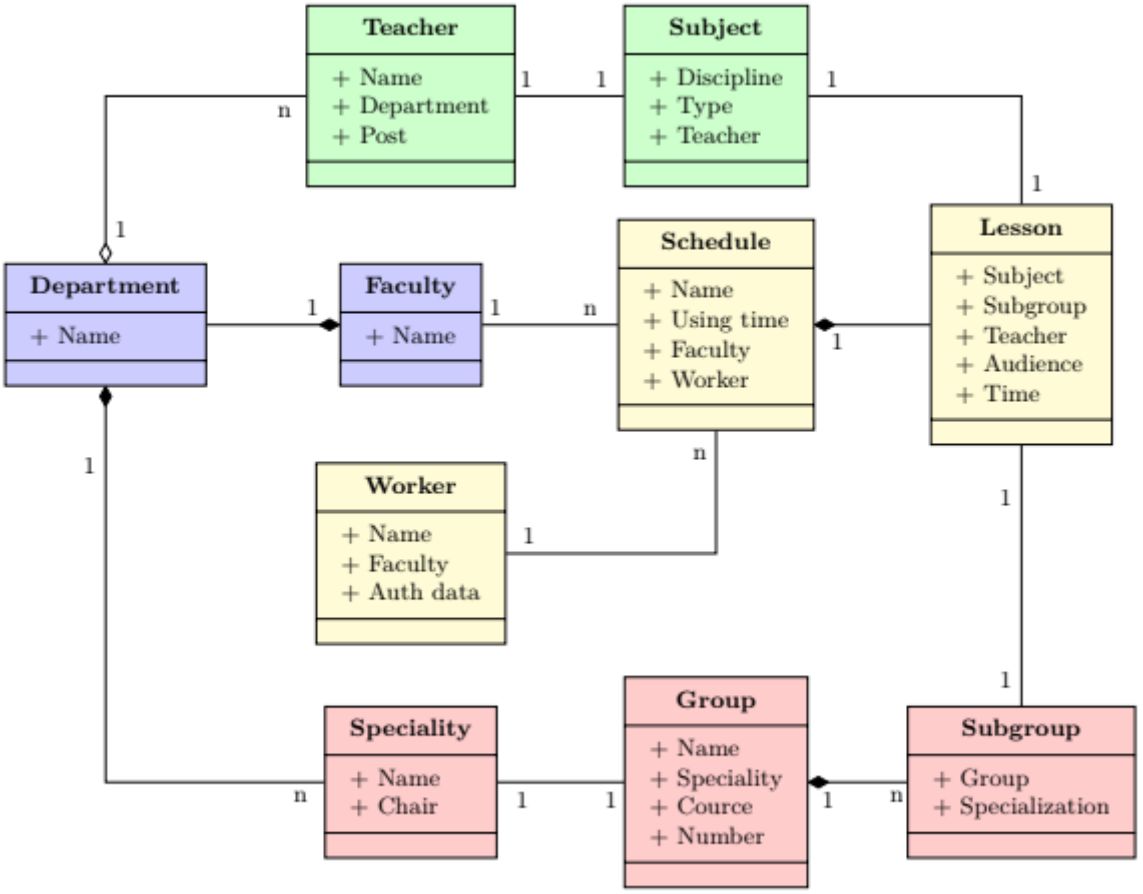

Рис. 2.1. Структура бази даних

З погляду програміста система повинна виглядати як постійне сховище об'єктів. Він може просто створювати об'єкти і працювати з ними, а вони автоматично зберігатимуться в реляційній базі даних.

3. РОЗДІЛ 3. ПРОЕКТУВАННЯ BACK-END ЧАСТИНИ

3.1. Клієнт-серверна архітектура веб-додатків

Архітектура клієнт-сервер є одним із архітектурних шаблонів програмного забезпечення та є домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Вона передбачає такі основні компоненти:

- набір серверів, які надають інформацію або інші послуги програмам, які звертаються до них;
- набір клієнтів, які використовують сервіси, що надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Загальноприйнятим є положення, що клієнти та сервери — це перш за все програмні модулі. Найчастіше вони знаходяться на різних комп'ютерах, але бувають ситуації, коли обидві програми — і клієнтська, і серверна, фізично розміщуються на одній машині; в такій ситуації сервер часто називається локальним.

3.1.1. Мікросервісна архітектура

Мікросервісна архітектура полягає в створенні для кожного з логічно відокремлених компонентів системи окремого модулю, пов'язаного з рештою.

Один з принципів проектування мікросервісних додатків додатків визначає, що розмір одного сервісу повинен бути таким, щоб повністю «вміщуватися» в голову програміста.

В рамках системи закладено низку модулів, частина з яких використовує у своїй роботі доступ до сервісів Google, зокрема Google Sheets та Google Calendar. При цьому для взаємодії посередництвом Google API потрібно пройти процедуру аутентифікації (рис. 3.1), закладену в методи бібліотек для основних платформ, в тому числі Node.js. Всі пакети мають відкритий вихідний код та поширюються разом з документацією.

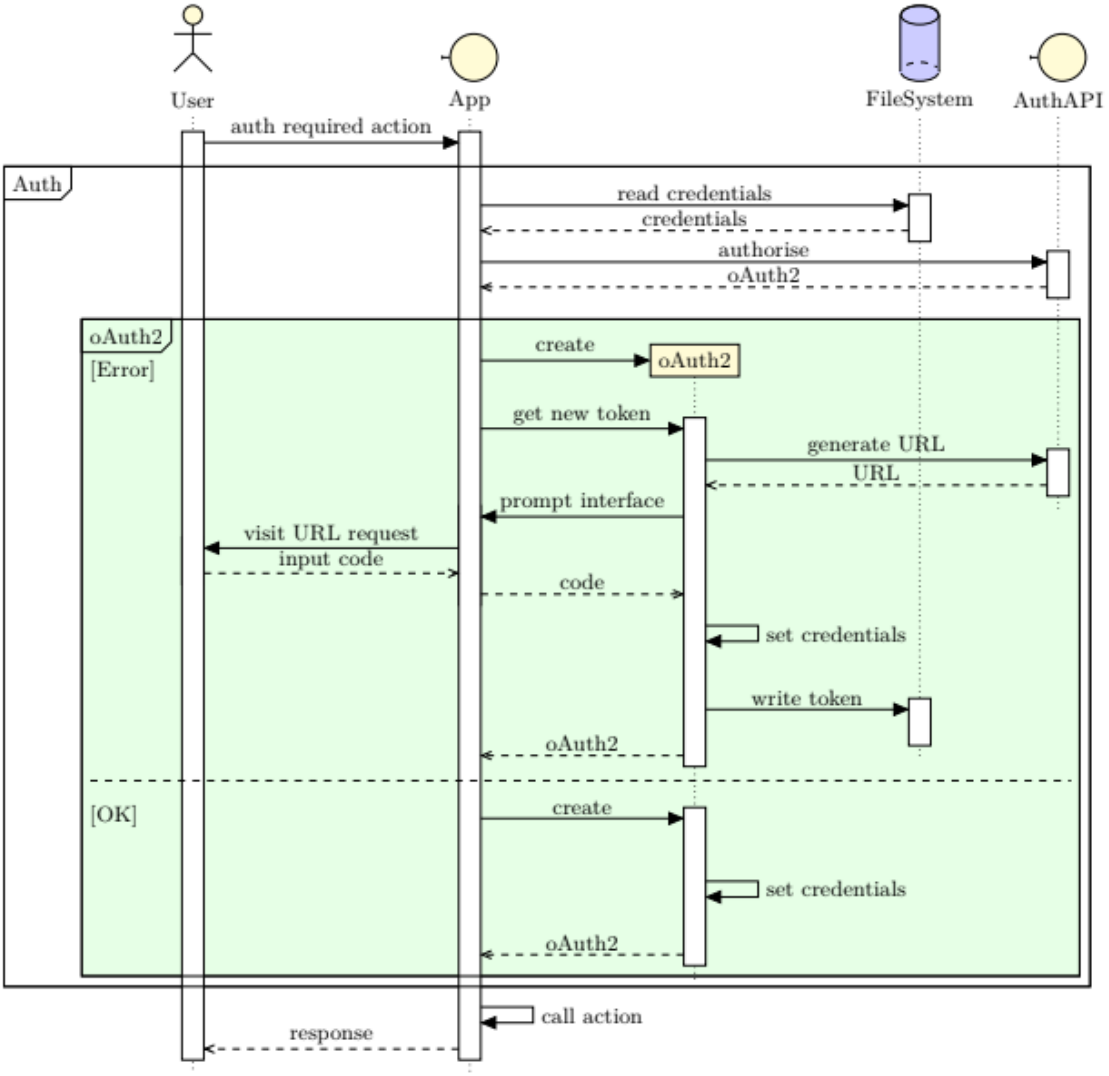

Рис. 3.1. Авторизація з сервісами Google

3.2. API веб-додатку

Прикладний програмний інтерфейс — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек тощо.

3.2.1. Rest api

REST — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм Філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування Всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

REST, як і кожен архітектурний стиль відповідає ряду архітектурних обмежень (англ. architectural constraints). Це гібридний стиль який успадковує обмеження з інших архітектурних стилів.[1]

Клієнт-сервер

Перша архітектура від якої він успадковує обмеження — це клієнт-серверна архітектура. Її обмеження вимагає розділення відповідальності між компонентами, які займаються зберіганням та оновленням даних (сервером), і тими компонентами, які займаються відображенням даних на інтерфейсі користувача та реагування на дії з цим інтерфейсом (клієнтом). Таке розділення дозволяє компонентам еволюціонувати незалежно.

Відсутність стану

Наступним обмеженням є те, що взаємодії між сервером та клієнтом не мають стану, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту.

Відсутність стану не означає що стану немає. Відсутність стану означає, що сервер не знає про стан клієнта. Коли клієнт, наприклад, запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта. Клієнт може залишити сторінку відкритою протягом кількох років, перш ніж натиснути посилання, і тоді сервер відповість на інший запит. Тим часом сервер може відповідати на запити інших клієнтів, або нічого не робити — для клієнта це не має значення.

Таким чином, наприклад дані про стан сесії (користувача, який автентифікувався) зберігаються на клієнті, і передаються з кожним запитом. Це покращує масштабовність, бо сервер після закінчення обробки запиту може звільнити всі ресурси, задіяні для цієї операції, без жодного ризику втратити цінну інформацію. Також спрощується моніторинг і зневадження, бо для того аби розібратись, що відбувається в певному запиті, досить подивитись лише на той один запит. Збільшується надійність, бо помилка в одному запиті не зачіпає інші.

3.2.2. SOAP API

SOAP — протокол обміну структурованими повідомленнями в розподілених обчислювальних системах, базується на форматі XML.

Спочатку SOAP призначався, в основному, для реалізації віддаленого виклику процедур (RPC), а назва була абревіатурою: Simple Object Access Protocol — простий протокол доступу до об'єктів. Зараз протокол використовується для обміну повідомленнями в форматі XML, а не тільки для виклику процедур. SOAP є розширенням мови XML-RPC.

SOAP можна використовувати з будь-яким протоколом прикладного рівня: SMTP, FTP, HTTP та інші. Проте його взаємодія з кожним із цих протоколів має свої особливості, які потрібно відзначити окремо. Найчастіше SOAP використовується разом з HTTP.

SOAP є одним зі стандартів, на яких ґрунтується технологія веб-сервісів.

3.3. JSON Web Token

Для забезпечення конфіденційності при обміні даними використовується JSON Web Token. Роути, що обробляють реєстраційні та авторизаційні запити, представлено на рис. 3.3.

JSON Web Token це стандарт токена доступу на основі JSON, стандартизованого в RFC 7519. Використовується для верифікації тверджень. JSON Web Token складається з трьох частин: заголовка, вмісту і підпису.

В корисному навантаженні зберігається будь-яка інформація, яку потрібно перевірити. Кожен ключ в корисному навантаженні відомий як «заява». Як і заголовок, корисне навантаження кодується в base64. Після отримання заголовку і корисного навантаження, обчислюється підпис.

3.4. Проектування та прототипування back-end частини

Для обміну інформацією між користувацьким додатком та back-end частиною використовується протокол передачі гіпертекстових даних HTTP. Передачу даних забезпечує стек транспортних протоколів TCP/IP.

Одним з способів побудови мереживих HTTP-додатків є використання асинхронного подієвого JavaScript–оточення Node. Для кожного з'єднання викликається функція зворотнього виклику, проте коли з'єднань немає Node засинає.

У Node не має функцій, що працюють напряму з I/O, тому процес не блокується ніколи. Як результат, на Node легко розробляти масштабовані системи [7].

Node широко використовує подієву модель, він приймає цикл подій за основу оточення, замість того, щоб використовувати його в якості бібліотеки. В інших системах відбувається блокування виклику для запуску циклу подій.

При розробці використано бібліотеку Express — гнучкий фреймворк для веб-застосунків, побудованих на Node.js, що надає широкий набір функціональності, полегшуючи створення надійних API.

Express забезпечує тонкий прошарок базової функціональності для веб-застосунків, що не спотворює звичну та зручну функціональність Node.js., при отриманні запиту він оброблюватиметься відповідно до визначення маршруту (рис. 3.2), де app є екземпляром express, METHOD є методом HTTP-запиту, PATH є шляхом на сервері, HANDLER є функцією-обробником, що спрацьовує, коли даний маршрут затверджено як співпадаючий.

```
1  app.METHOD(PATH, HANDLER)
2
```

Рис. 3.2. Структура визначення маршрутів

3.4.1. Роути

В процесі роботи використовується протокол прикладного рівня HTTP. Обмін повідомленнями йде за схемою «запит-відповідь». Для ідентифікації ресурсів HTTP використовує URI.

В додатку обробляються основні HTTP методи для взаємодії об’єктами (рис. 3.3).

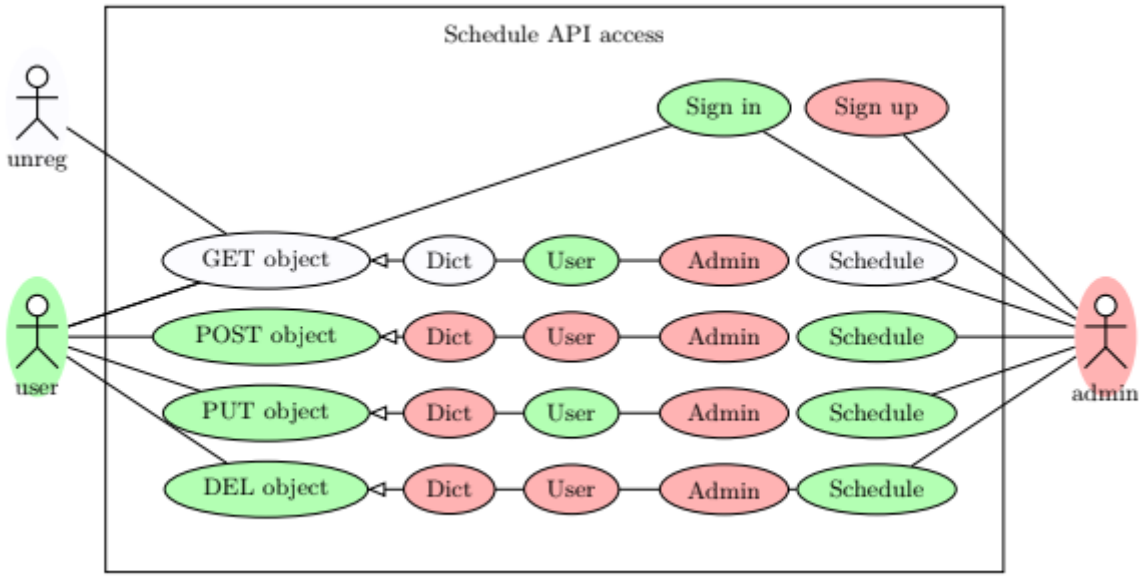

Рис. 3.3. Доступ на виконання запитів до системи

Протокол HTTP не зберігає свого стану між парами «запит-відповідь». Компоненти, що використовують HTTP, можуть самостійно здійснювати збереження інформації про стан, пов'язаний з останніми запитами та відповідями.

Одним з розповсюджених способів реалізації цього можна назвати так звані *cookies* — невеликі записи, що зберігаються браузером. Зазвичай, вони встановлюються при виконанні користувачем певних дій та надсилаються серверу разом з наступними запитами.

На рис. 3.4 зображено процес створення адміністратором нового користувача системи. Ця функція може викликатися після відповідного запиту з компоненту веб-додатку, котрий детально описано в пункті 4.3.3.

Після отримання зазначеного POST запиту, сервер перевірить, чи має користувач відповідні права (блок Auth рис. 3.7), створить об’єкт користувача з відповідними правами та збереже його в базі даних.

Створений користувач може входити до системи (рис. XX) для виконання певних задач з користування системою.

Можна зазначити, що адміністратор теж є користувачем, проте з особливими правами (диференціація на рис. 3.3).

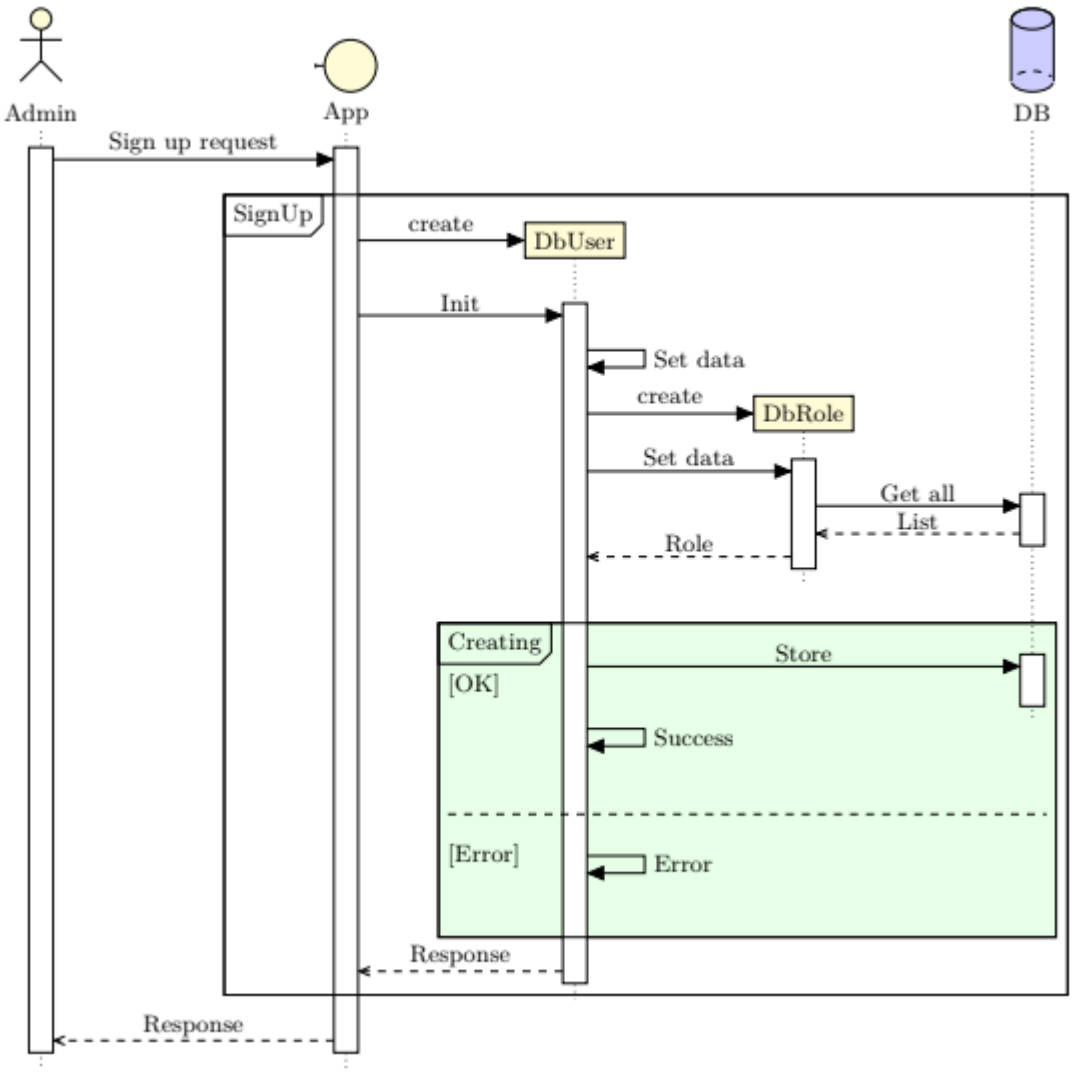

Рис. 3.4. Процес створення адміністратором нового користувача системи

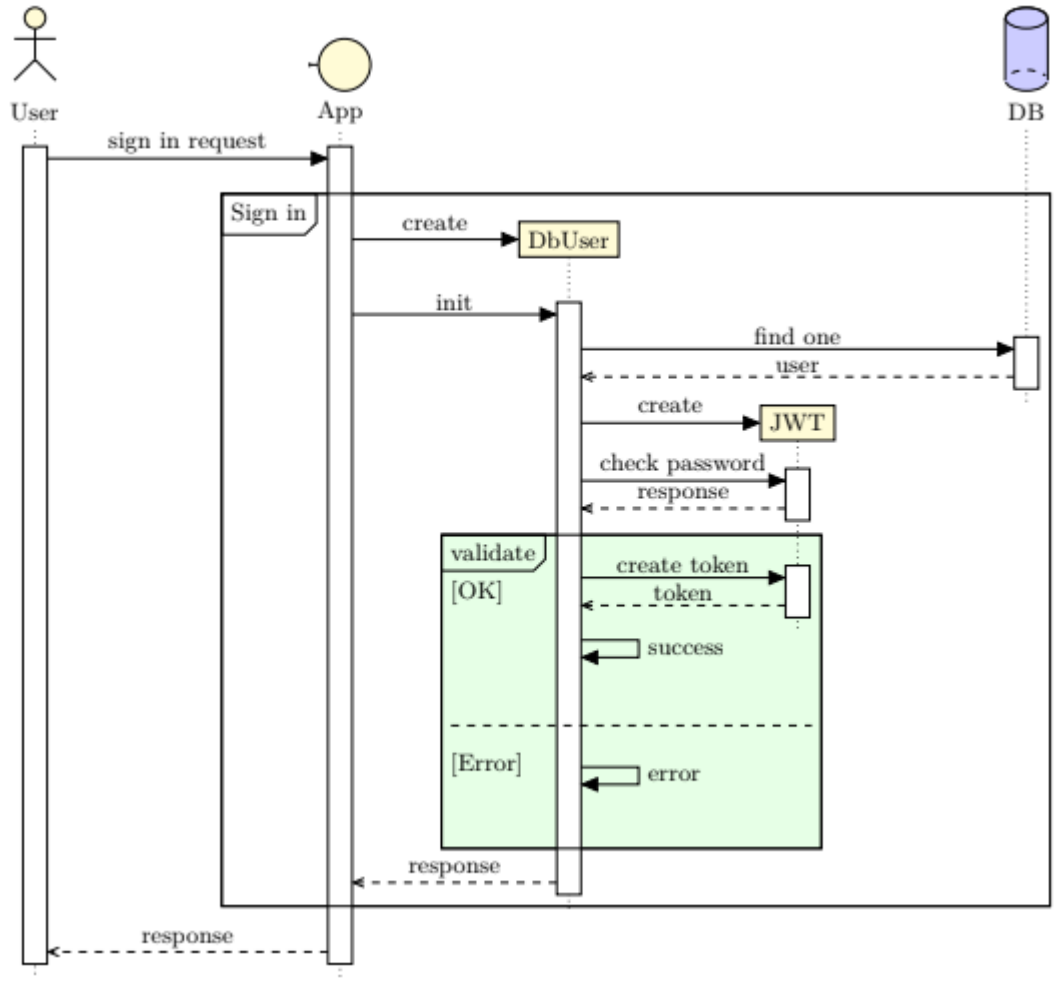

Рис. 3.5. Вхід користувача до системи

3.4.2. Моделі

В процесі проектування закладено серію моделей, що відповідають об’єктам предметної області. В рамках системи зберігаються в базі даних у вигляді таблиць з певними взаємозвязками (реляційну модель описано в підрозділі 2.2). Для доступу до даних використовуються основні HTTP методи, що відповідають операціям CRUD, їх перелічено нижче.

1. GET. Запитує вміст вказаного ресурсу, який може приймати параметри, що передаються в URI (рис. 3.6). Згідно зі стандартом, ці запити є ідемпотентними — багатократне повторення одного і того ж

запиту GET приводить до однакових результатів (за умови, що сам ресурс не змінився за час між запитами).

В запропонованій реалізації запит GET має дві версії — з параметром (ID) та без нього. Останній виконує дію (надає користувачу) не до конкретного об’єкту, а до всієї множини, що є необхідним в певних ситуаціях (наприклад, відображення списку всіх викладачів за певним критерієм).

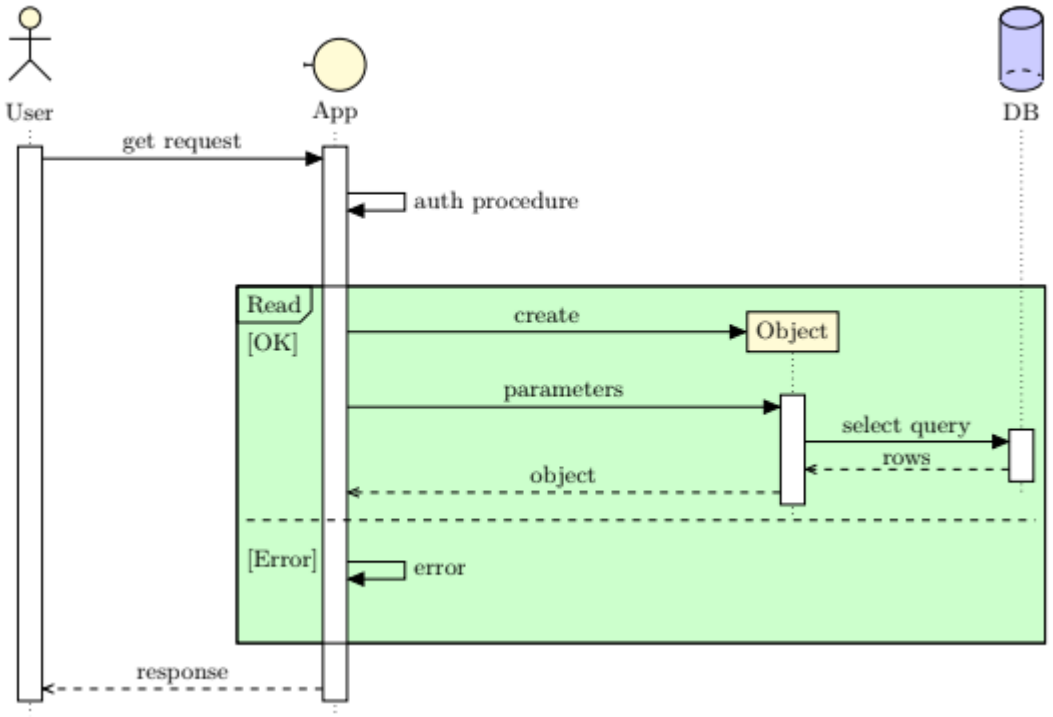

Рис. 3.6. Виконання запиту на отримання об’єкта

2. HEAD. Аналогічний GET, за винятком того, що у відповіді сервера відсутнє тіло. Це може бути необхідно для отримання мета-інформації.

3. POST. Передає дані (наприклад, з форми на веб-сторінці) заданому ресурсу. При цьому передані дані включаються в тіло запиту. На відміну від методу GET, метод POST не є ідемпотентним, тобто багатократне повторення одних і тих же запитів POST може повертати різні результати (рис. 3.7).

   На першому етапі відбувається перевірка доступу користувача это створення об’єкту цього типу (авторизація), відповідно до прав доступу (рис. 3.3).

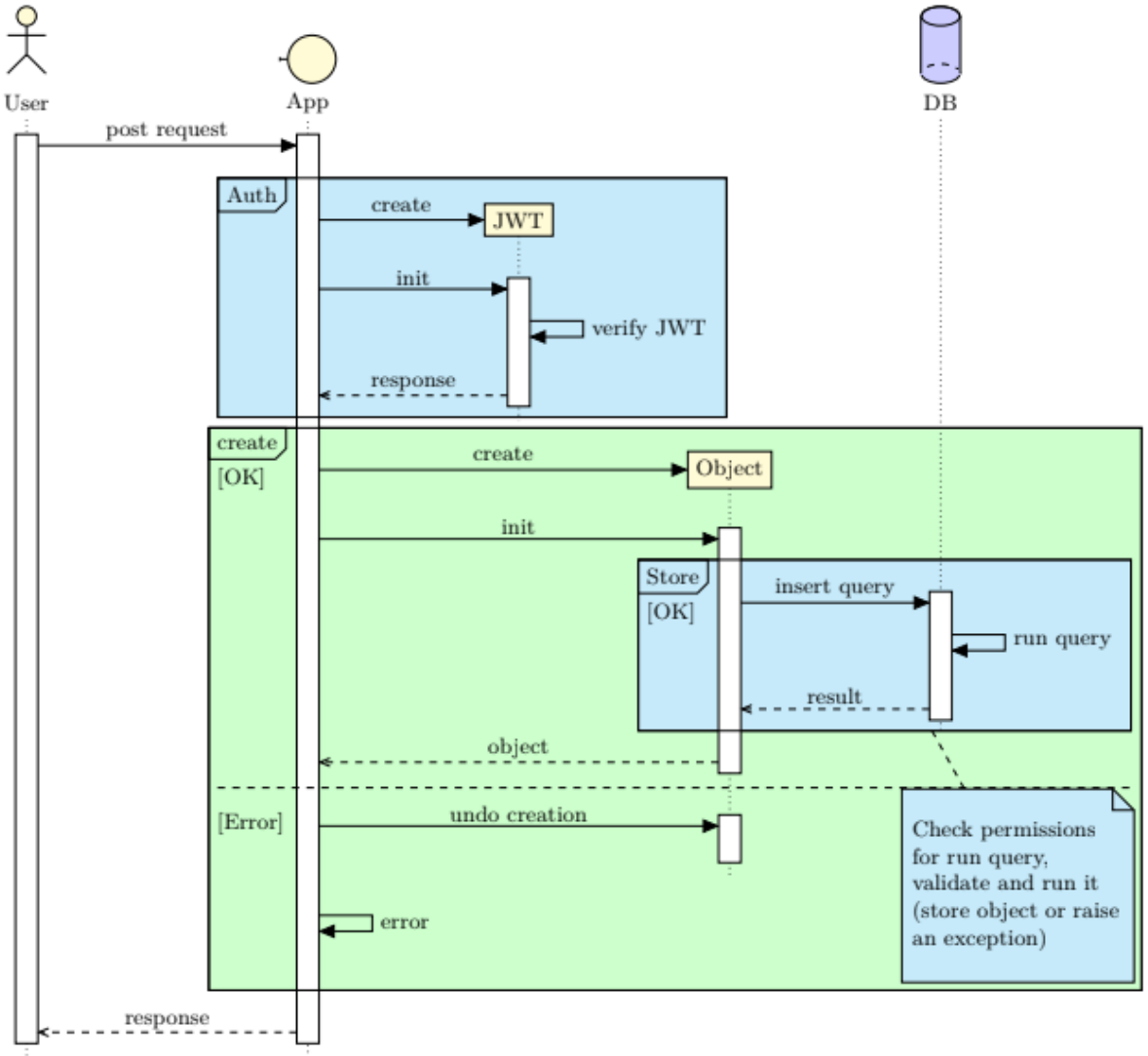

Рис. 3.7. Виконання запиту на створення з аутентифікацією

4. PUT. Завантажує вказаний ресурс на сервер. В розроблюваній системі використовується для редагування існуючих даних (рис. 3.8). В процесі виконання, спочатку з бази даних силами ORM вибирається конкретний об’єкт, в нього вносяться зміни, після чого він записується до сховища на заміну попередньої версії.
5. PATCH. Завантажує частину ресурсу на сервер. При розробці необхідності у використанні не знайдено.

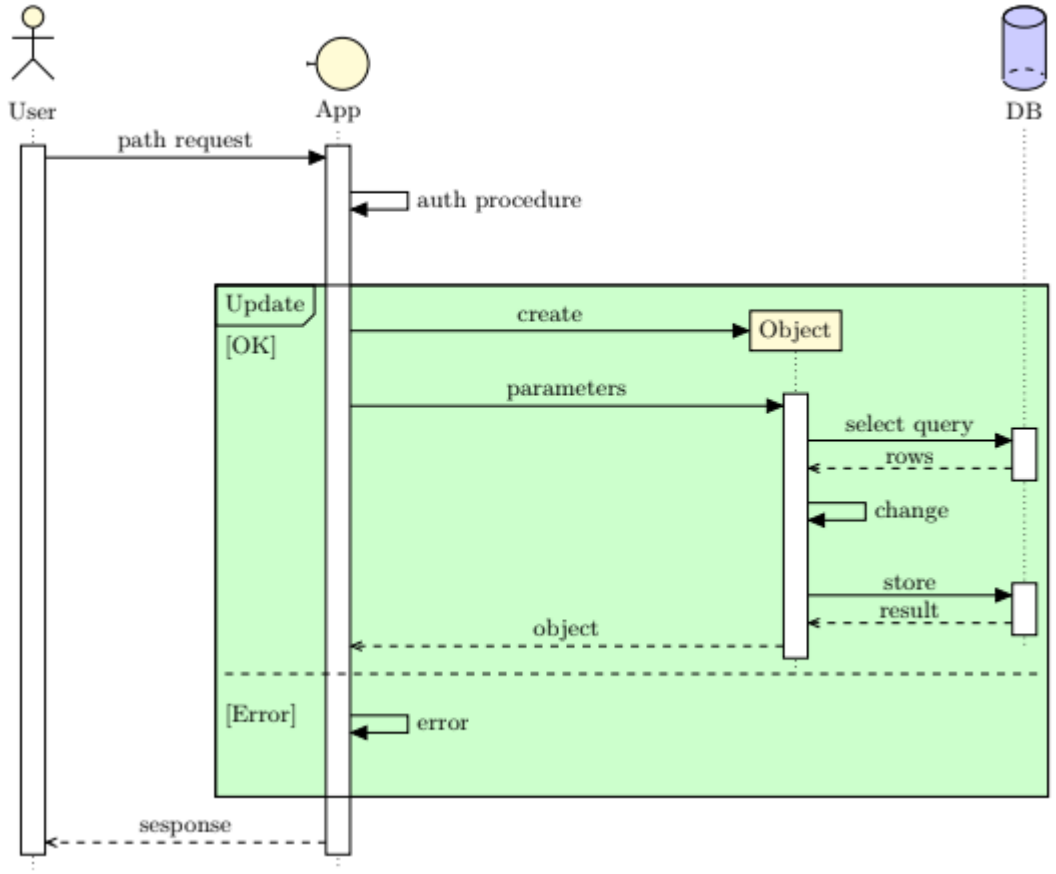

Рис. 3.8. Виконання запиту на модифікацію існуючого об’єкту

6. DELETE. Видаляє вказаний ресурс.

Слід звернути увагу, що в процесі виконання запиту на видалення об’єкту в системі, видалення як такого не відбувається. Замість цього в окреме поле таблиці вноситься інформація про час виконання цієї процедури (рис. 3.9).

Такий спосіб реалізації дозволяє з однієї сторони приховати дані, відмічені як видалені від подальшого використання, а з іншої — зберегти їх там, де вони вже використовуються. В іншому випадку, у зв’язку з реляційністю бази потрібно було б вирішувати дилему — або проводити циклічне видалення для збереження цілісності даних, втрачаючи всі об’єкти, що посилаються на той, що видаляється; або ускладнювати структури даних, що потенційно призведе до дублювання даних.

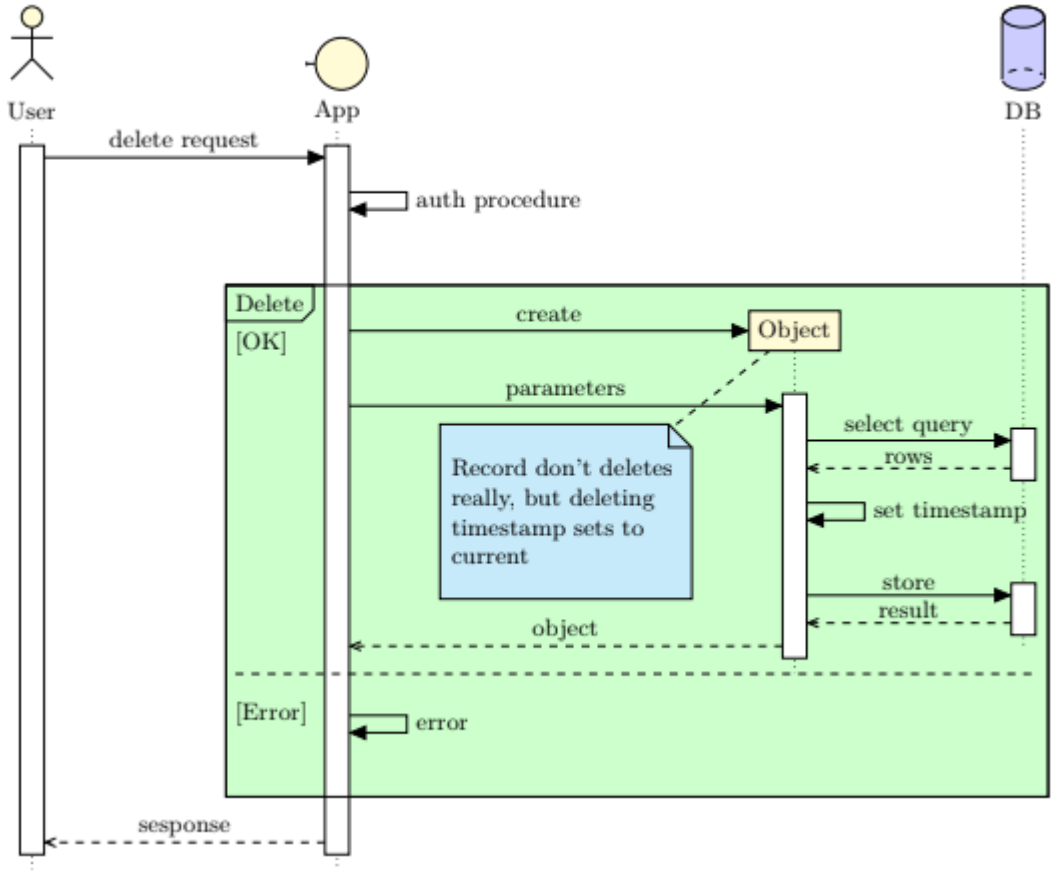

Рис. 3.9. Виконання запиту на видалення об’єкту

3.4.3. Публічне API

В процесі проектування створено структуру роутів, котра може використовуватися сторонніми сервісами, у тому числі — і без авторизації в системі, що дозволяє отримувати інформацію про розклади власними силами для подальшого використання тим чи іншим чином.

Також було проаналізовано перспективи при використанні QR-кодів (рис. 3.10) з метою супроводження традиційного паперового розкладу (та інших документів), що публікується на стендах. Хоча термін «QR code» є зареєстрованим товарним знаком японської корпорації «DENSO Corporation», їх використання не обкладається ніякими ліцензійними відрахуваннями, коди описані та опубліковані як стандарти ISO [1]. Основна перевага QR-коду – легке розпізнавання скануючим обладнанням (за допомогою мобільного телефону, планшета або ноутбука з камерою, на яких встановлена програма для зчитування кодів, тощо).

Одним з способів використання QR-кодів в навчальному процесі, крім запропонованих (зокрема, задля забезпечення швидкого доступу до навчально-методичного забезпечення, довідкової літератури, веб-сервісів навчального закладу) [1, с. 146], можна назвати надання доступу до електронної версії розкладу.

Рис. 3.10. Приклад QR-коду з посиланням

4. РОЗДІЛ 4. ПРОЕКТУВАННЯ FRONT-END ЧАСТИНИ

4.1. Аналіз існуючих бібліотек для розробки SPA

На сьогодні, одними з росповсюджених фреймворків для розробки SPA (single page applications) є React, Angular та Vue (рис. 4.1).

Рис. 4.1. Зліва направо: React, Angular, Vue

Angular - Javascript-фреймворк, створений на основі TypeScript. Розроблений і підтримуваний компанією Google, він описується як JavaScript MVW-фреймворк. На даний момент останньою версією є 4. Фреймворк Angular використовується такими компаніями, як Google, Wix, weather.com, healthcare.gov і Forbes.

Vue - ще один JS-фреймворк. Творці Vue описують його як «інтуїтивно зрозумілий та швидкий, призначений для створення інтерактивних інтерфейсів». Вперше він був представлений колишнім співробітником компанії Google Еваном Ю (Evan You) в лютому 2014 року. На даний момент фреймворк використовується такими компаніями, як Alibaba, Baidu, Expedia, Nintendo, GitLab.

4.2. React

При розробці мобільного додатку використано фреймворк React Native, в основі якого знаходиться бібліотека React, призначена для створення користувацьких інтерфейсів. На відміну від React, призначеного для розробки односторінкових веб-додатків, React Native спрямований на мобільні платформи.

React — відкрита JavaScript бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. React використовують Facrbook, Airbnb, Uber, Netflix, Twitter, Pinterest, Reddit, Udemy, Wix, Paypal, Imgur, Feedly, Stripe, Tumblr, Walmart та інші.

React дозволяє розробникам створювати великі веб-додатки, які використовують дані, котрі змінюються з часом, без перезавантаження сторінки. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер (MVC) і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як AngularJS. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків.

React підтримує віртуальний DOM, а не покладається виключно на DOM браузера. Це дозволяє бібліотеці визначити, які частини DOM змінилися, порівняно зі збереженою версією віртуального DOM, і таким чином визначити, як найефективніше оновити DOM браузера. Як бібліотека інтерфейсу користувача React часто використовується разом з іншими бібліотеками, такими як Redux, проте у його використанні при розробці проекту не було необхідності.

React надає розробникам безліч методів, які викликаються під час життєвого циклу компонента (рис. 4.6), вони дозволяють нам оновлювати UI і стан додатку. Коли необхідно використовувати кожен з них, що необхідно робити і в яких методах, а від чого краще відмовитися, є ключовим моментом до розуміння як працювати з React.

Конструктори є основною ООП — це спеціальна функція, яка буде викликатися щоразу, коли створюється новий об'єкт. Важливо викликати функцію *super* в випадках, коли наш клас розширює поведінку іншого класу, який має конструктор. Виконання цієї спеціальної функції буде викликати конструктор нашого батьківського класу і дозволяти йому проініціалізувати себе.

Конструктори — це відмінне місце для ініціалізації компонента — створення будь-яких полів (змінні, що починаються з *this.*).

Це також єдине місце де слід встановлювати стан безпосередньо перезаписуючи поле *this.state*. У всіх інших випадках необхідно використовувати *this.setState*.

За замовчуванням, всі компоненти будуть перемальовувати себе всякий раз, коли їх стан змінюється, змінюється контекст або вони приймають *props* від батьківського компонента. Якщо перерисовка компонента досить важка (наприклад генерація графіка), то у розробників є доступ до спеціальної функції, яка дозволяє контролювати цей процес.

4.3. Проектування та прототипування front-end частини

4.3.1. Webpack

При створенні сайту досить стандартною практикою є мати певний процес збірки на місці, щоб полегшити розробку і підготовку файлів до роботи.

Можливо використовувати Grunt або Gulp, побудувавши ланцюжки перетворень, які дадуть можливість подати код в один кінець ланцюжка і отримати мінімізовані CSS та JavaScript на іншому.

Подібні інструменти розробки досить популярні і корисні в наші дні. Проте, є й інший метод полегшення розробки — Webpack.

Webpack є так званим «збиральником модулів». Він приймає модулі JavaScript, аналізує їх залежності один від одного, а потім з'єднує їх найефективнішим способом, випускаючи у кінці лише один JavaScript файл. З Webpack, модулі не обмежені тільки файлами JavaScript. Завдяки частині *loaders*, Webpack розуміє, що модуль JavaScript може потребувати CSS файл, а цей CSS файл може потребувати зображення. Результат роботи Webpack буде містити тільки те, що потрібно у проекті.

4.3.2. Babel

Нажаль, при постійному розвитку мов програмування невід'ємною є ситуація, що реалізація часто відстає від специфікації. Більш того, різні реалізації по-різному відстають від специфікації. Написавши код, ми не можемо гарантувати, де він буде запускатися, а де - ні.

Виходячи з цього можна зробити висновок, що потрібно писати код, дотримуючись старих стандартів. На щастя, є інший шлях: ми можемо писати код з використанням всіх найновіших можливостей, але перед публікацією автоматично транслювати його (тобто переводити з одного виду в інший) в стару версію.

Сама природа JS і його способи використання готують нас до того, що ніколи не настане моменту, коли у всіх користувачів буде остання версія інтерпретатору. Люди використовували і продовжать використовувати різні браузери і різні версії браузерів, різні версії Node.js і так далі. Використання нових синтаксичних конструкцій в такій ситуації практично

неможливо. Запуск коду на платформі що не підтримує новий синтаксис призведе до синтаксичну помилку.

Закономірним вирішенням цієї проблеми стала поява Babel - програми, яка бере вказаний код і повертає той же код, але транслювали в стару версію JS. Фактично, в сучасному світі Babel став невід'ємною частиною JS. Всі нові проекти так чи інакше розробляють з його використанням.

4.3.3. Розробка компонентів

В термінах React, всі частини-відображення іменуються компонентами. В роботі спроектована серія компонентів для різних частин системи та розроблено прототипи деяких з них.

На рис. 4.2 зображено компонент для створення адміністратором нового користувача системи. Слід наголосити, що кожен компонент є окремою частиною і тому можливий для використання у подальшому в інших системах при виконанні певних вимог (так зване «повторне використання»).

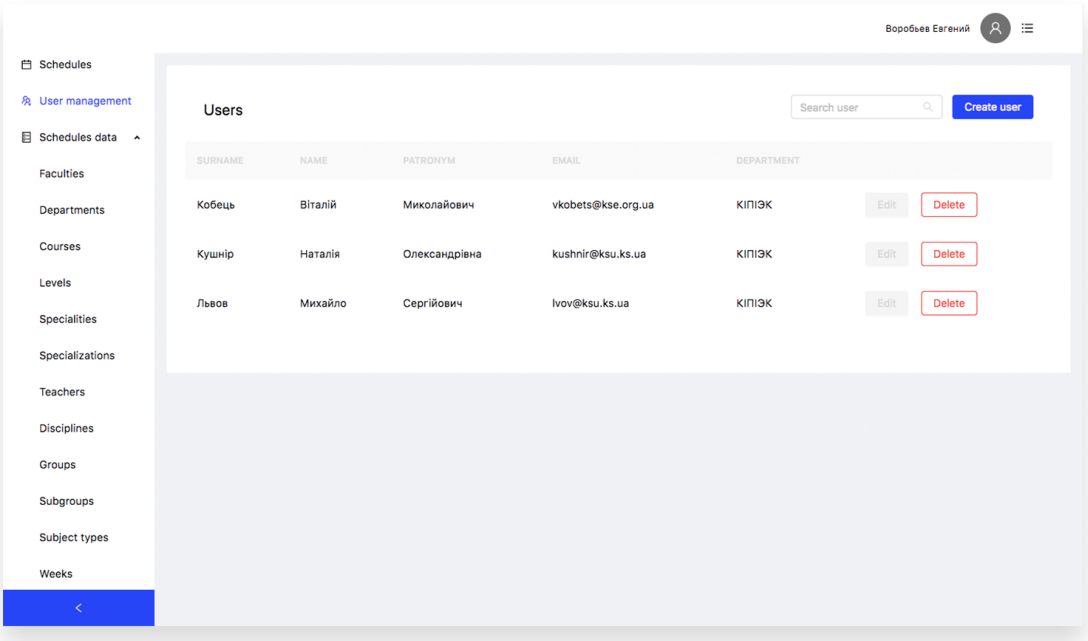

Рис. 4.2. Адміністративна панель (керування користувачами)

На приведеному вище зображенні натискання на кожну з кнопок призводить до виклику відповідного методу API шляхом надсилання певного HTTP запиту.

Перед доступом до адміністративної панелі адміністратору необхідно авторизуватися у системі, в результаті чого буде створено і збережено силами його веб-браузеру JWT (підрозділ 3.3). Після цього, якщо він має відповідні права (рис. 3.3) та верифікація токену пройшла успішно (блок Auth на рис. 3.7), йому буде відображена відповідна панель.

Слід зауважити, що подібний процес перевірки відбувається при виконанні кожного запиту, крім тих, що не потребують авторизації (доступні для незареєстрованих користувачів, рис. 3.3).

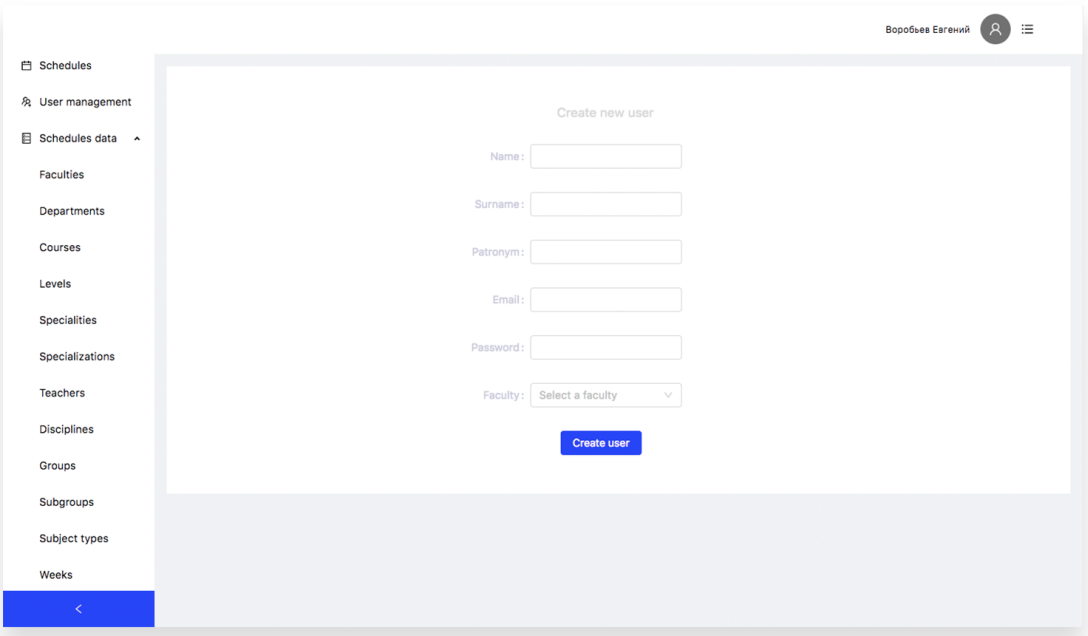

Рис. 4.3. Створення нового користувача системи

Перша прерогатива адміністратора системи — створення нових користувачів. При цьому відбувається вищеописана процедура, та реалізується процес, котрий детально зображено на діаграмі 4.3.

Певна частина об’єктів системи може вважатися більш-менш константною, це структура закладу вищої освіти, окремі словникові дані (зокрема, назви посад професорсько-викладацького складу та види занять).

Окремо можна відзначити використання об’єктів часу (номера занять впродовж дня). В спроектованій системі одним з можливих шляхів доступу до розкладу є його експорт до сервісу Google Calendar в серію календарів. В подальшому, потенційні користувачі можуть підписуватися на оновлення відповідних календарів (зокрема, груп та викладачів) для отримання актуальної інформації в довільний момент часу.

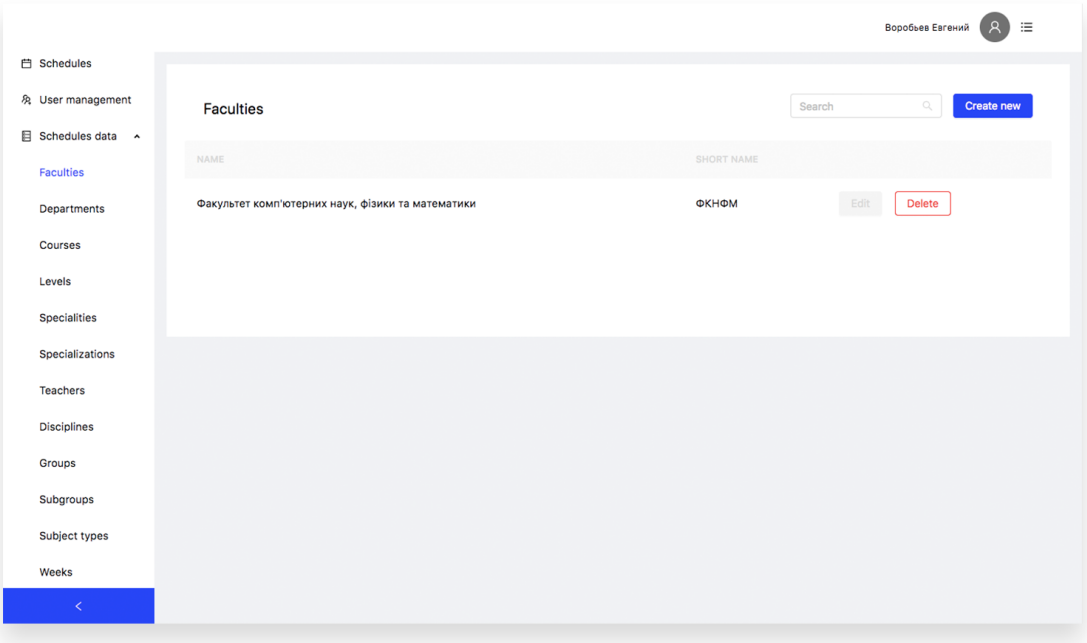

Рис. 4.4. Керування факультетами

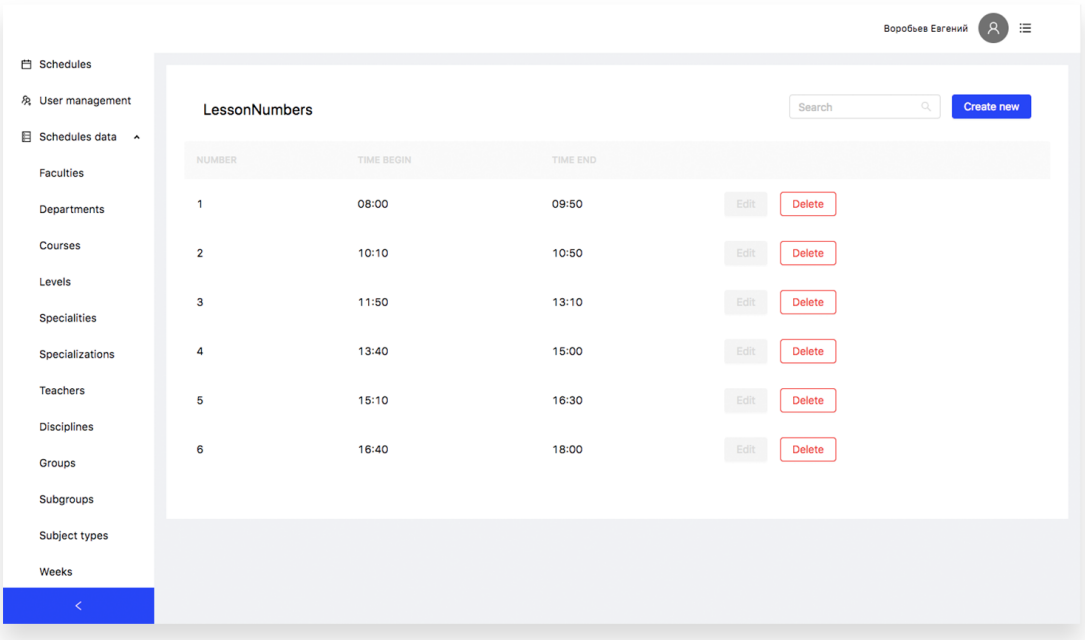

Рис. 4.5. Керування часом занять

4.4. Менеджери стану

Redux — це інструмент управління як станом даних, так і станом інтерфейсу в JavaScript-додатках. Він підходить для односторінкових додатків, в яких управління станом може з часом стає складним. Redux не пов'язаний з якимось певним фреймворком, і хоча розроблявся для React, може використовуватися з Angular або jQuery.

С Redux всі компоненти отримують свої дані зі сховища. Також зрозуміло, куди компонент повинен відправити інформацію про зміну стану — знову ж в сховище. Компонент тільки ініціює зміну і не піклується про інших компонентах, які повинні отримати цю зміну. Таким чином, Redux робить потік даних більш зрозумілим.

Загальна концепція використання сховищ для координації стану програми — це шаблон, відомий як Flux. Цей шаблон проектування доповнює односпрямований потік даних як в React.

Redux використовує тільки одне сховище для всього стану програми. Оскільки стан знаходиться в одному місці, його називає єдиним джерелом істини. Структура даних сховища повністю залежить від вас, але для реального застосування це, як правило, об'єкт з декількома рівнями укладення.

Такий підхід єдиного сховища є основною відмінністю між Redux і Flux з його численними сховищами.

Згідно з документацією Redux, «Єдиний спосіб змінити стан - передати *action* — об'єкт, що описує, що сталося». Це означає, що програма не може безпосередньо змінити стан. Замість цього, необхідно передати *«action»*, щоб висловити намір змінити стан в сховищі (рис. 4.6).

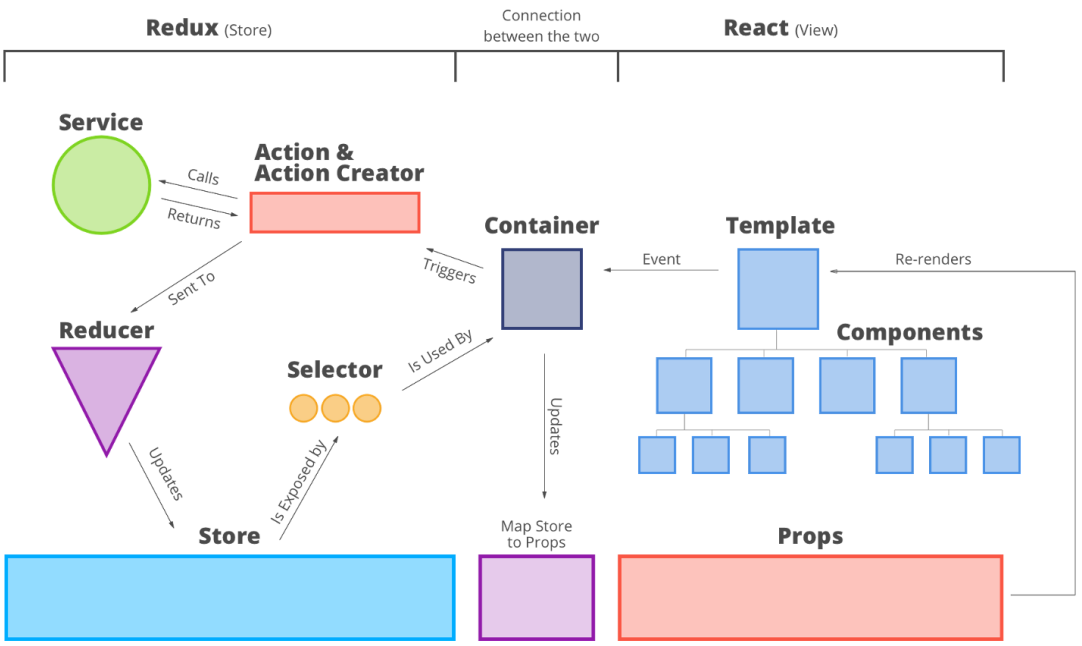

Рис. 4.6. Взаємодія Redux та React

ВИСНОВКИ

Для виконання поставлених завдань було проведено аналіз характеристики існуючих систем планування, зокрема обсяг їх можливостей. При підготовці до проектування було приділено увагу окремим частини процесу підготовки розкладу на прикладі факультету комп'ютерних наук, фізики та математики ХДУ.

На основі проведеного аналізу розроблено базові вимоги щодо можливостей додатку та його інтерфейсу.

Суттєву частку роботи приділено аналізу існуючих технологій всіх рівнів для створення веб-додатків. Детально досліджено роботу клієнт-серверних додатків та проектуванню API.

Відповідно до створених вимог розробити проект додатку та бекенд частини. Розробити робочий прототип бекенд частини (зокрема реалізувано структуру бази даних засобами PostgreSQL, моделі з використанням ORM Squalize та окремі частини API і інтерфейсу додатку.

Сформовано проект документації до публічного API. При написанні ключових частин використано спеціальну форму коментарів, що забезпечують інтеграцію опису функцій та їх параметрів в підказки популярних IDE (інтегрованих середовищ розробки). Останнє є корисним при подальшій розробці, особливо при використанні існуючої кодової бази сторонніми розробниками, що є цілком можливим, зважаючи на модульність проекту при використанні мікросервісної архітектури.

При розробці проекту використовується система контролю версій git з публічним репозиторієм на сервісі GitHub (github.com/Rembut/gCalShedule), що дозволяє використовувати сучасні методи сумісної роботи та, одночасно з тим, дозволяє використовувати результати проведеного дослідження всім охочим під ліцензією MIT.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Воронкін О. С. Можливості використання системи QR-кодів у вищій школі //Матеріали четвертої міжнародної науково-практичної конференції FOSS Lviv 2014. – 2014. – С. 145-149.
2. ДСТУ ГОСТ 7.1:2006 "Система стандартів з інформації, бібліотечної та видавничої справи. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання".
3. Борисов В. В., Литвинов А. С. Використання сучасних автоматизованих систем управління навчальним закладом для складання розкладу. – 2015.
4. Ситник Н. В. Проектування баз і сховищ даних: Навч. посібник //К.: КНЕУ. – 2004.
5. Codd E. F. A relational model of data for large shared data banks //Communications of the ACM. – 1970. – Т. 13. – №. 6. – С. 377-387.
6. Дейт К. Д. Введение в системы баз данных. – Вильямс, 2008.
7. Zeiss M. Node.js v0.12.5 Manual & Documentation / Mirco Zeiss., 2015. – 242 с.