ВЕБ-ОРІЄНТОВАНИЙ ДОДАТОК ДЛЯ ВІДСТЕЖЕННЯ ЛАБОРАТОРНИХ РОБІТ

“TASKER”

*Гайдуков Ростислав В’ячеславович*, Україна

Чернівецький національний університет імені Юрія Федьковича

Факультет математики та інформатики

[rostyslav.haidukov@live.com](mailto:rostyslav.haidukov@live.com)

В навчального процесі часто виникають багато труднощів пов’язаних з організаційними питаннями. Багато завдань і лабораторних робіт – викладачу важко стежити, що вже виконав студент, що ще ні і яке у нього завдання взагалі. Це призводить до великої кількість паперової роботи. Це не є добре, бо замість того, щоб думати про навчальний процес викладачу доводиться думати про те, хто які лабораторні здав/не здав. Так як ми живемо в епоху інформаційних технологій, то чому б не автоматизувати цей рутинний процес?

Велику популярність набувають веб-додатки, які надають досвід користувача близький до користування звичайними Desktop-додатками. Це не дивно, тому що веб технології активно розвиваються і залишають свої недоліки в минуле. Все більше додатків тепер можуть похизуватися web-версією, і в той же час існуючі веб-додатки переносять більше логіки на клієнтську сторону – на Front-End.

Тому я вирішив розробити такий додаток, так званий Single-Page application, тобто насправді у нас, на відміну від традиційних веб-сайті, є один невеликий html файл і багато JavaScript коду, який буде керувати всім функціоналом і динамічно міняти нашу розмітку.

Додаток дозволяє створити певний Workspace, долучити туди студентів і слідкувати за прогресом студентів у виконанні завдань.

При розробці Single-Page додатків доводиться монолітний додаток розбивати на 2 окремі додатки: клієнтський і серверний. Спілкуватися вони будуть по REST. REST – такий архітектурний стиль обміну ресурсів між додатками, який базується на протоколі HTTP і накладає ряд обмежень:

1. Модель клієнт-сервер.
2. Stateless (Відсутність стану додатку на сервері. Уся інформація про стан додатку повинна знаходитися на клієнті).
3. Підтримка кешування.
4. Уніфікований інтерфейс (Всі ресурси ідентифікуються в запитах, всі ресурси концептуально відрізняються від представлення, які повертаються клієнтам. Зазвичай для представлення ресурсів використовується формат JSON).
5. Шари абстракції.

Так як додаток stateless, то стан додатку (дані про поточного користувача в системі, авторизаційні дані) зберігається на клієнті. В HTML 5 є WebStorage API, яке дозволяє зберігати дані на клієнті. Я використовую об’єкт LocalStorage, яке належить до наданої специфікації. Він дозволяє зберігати об’єкти у вигляді пар ключ – значення. Для авторизації використовується технологія JSON Web tokens. Це стандарт для створення токенів доступа, які підписуються секретний ключом, відправляються на клієнт. Клієнт повинен в кожний HTTP запит на захищений ресурс включати такий header:  
Authorization: Bearer {token}. Сервер повинен, маючи секретний ключ, перевіряти валідність токена.

Для розробки Front-End використав добре відому бібліотеку React. Вона позволяє декларативно описувати складні інтерфейси веб та мобільних додатків. За рахунок використання технології Virtual DOM при будь-яких змінах в інтерфейсі перемальовуватися буде тільки те, що потрібно, а не ціла сторінка. Це суттєве спрощує розробку додатків. Також використані такі технології: Webpack (утиліта, яка збирає JavaScript код та необхідні ресурси), Babel (утиліта, яка дозволяє трансформувати JavaScript код, написаний з використанням сучасних можливостей мови, в більш старші стандарти мови для сумістності з старими браузерами), Axios (Бібліотека, що надає крос-браузерний та кросплатформенний (браузер, Node.js) Promise-based API для роботи з HTTP запитами), material-ui (бібліотека React компонентів під популярний зараз Google Material Design).

Якби ми не переносили логіку на клієнт, але нам все рівно потрібно десь зберігати дані, нам потрібне сервер з базою даних. В якості СКБД я обрав MongoDB. Це нереляційна документоорієнтована СКБД. Вона класифікує себе як NOSQL, використовує JSON-подібні документи для схеми даних. На відміну від традиційних СКБД, таких як MySQL чи PostgreSQL, тут не використовується табличний спосіб представлення і не пітримується мова запитів SQL. від Основні її переваги:

* Простота розробки, так як відсутня жорстко задана схема.
* Динамічні і гнучкі запити.
* Швидкість обробки даних.

Основним недоліком є обмежені можливості по забезпеченню консистентності даних, але так як у моєму додатку немає великої кількості відношень між об’єктами предметної області, то це не є проблемою, а гнучкість JSON формату значно пришвидшила швидкість розробки.

Серверну частину (Back-End) реалізував на Node.js. Це середовище JavaScript, яке надає неблокуючу асинхронну модель для обробки вхідних даних. В основі лежить движок JavaScript V8 (той самий, що використовується web-браузером Google Chrome) та C++ бібліотека libuv вона надає можливість з JavaScript коду звертатися до мережі та файлової системи, виконувати паралельні задачі тощо. Принциповою відмінністю серверних додатків написаних на Node.js від додатків написаних на класичних технологіях таких як Java, PHP є неблокуючий ввід-вивід. Візьмемо для прикладу серверний застосунок, написаний на Java. Кожний клієнтський запит обробляється в окремому потоці, тобто потрібні ресурси на створення нового потоку і подальшої синхронізації. Операції вводу-виводи – блокуючі. Це означає, що коли ми робимо запит в базу даних, чи читаємо з файлу нам доводиться чекати завершення результату і не можемо нічого робити в даному потоці. Неблокуючий ввід/вивід означає, що ми використовуємо тільки один потік, а операції вводу/виводу виконуємо асинхронно – відправляє підписуємося на результат операції і йдемо виконувати інші завдання і в момент закінчення асинхронної операції ставимо її в чергу завдань і коли у нашого потоку не буде активних завдань він по черзі буде виконувати завдання з черги. Це додає приросту до продуктивності, так як наші ресурси не простоюють. З недоліків: наш код втрачає лінійну структуру і стає важчим у підтримці, але останні можливості мови JavaScript, такі як конструкція async/await успішно вирішує дану проблему. Також, можуть виникнути проблеми, якщо в чергу попаде задача, яка потребує важких обчислень. Так як потік один, то він заблокується. Саме тому Node.js зазвичай використовується, коли потрібно виконувати багато паралельних асинхронних, але простих задач. В останні версії Node.js тепер включені засоби для багатопотоковості такі як Web Workers, але загалом інфраструктура ще сира. Для роботи з MongoDB використав ORM Mongoose. ORM – це бібліотека, яка позволяє відобразити сутності бази даних у об’єкти/класи мови програмування, що підтримує ООП.

<https://nodejs.org/uk/>

<https://reactjs.org/>

<https://developer.mozilla.org>

<https://www.mongodb.com/>

<https://webpack.js.org/>

<https://babeljs.io>

<https://expressjs.com/>

<https://mongoosejs.com/>

<https://en.wikipedia.org/wiki/Representational_state_transfer>

<https://jwt.io/>

<https://material.io/design/>