Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій факультету інформатики

ПОБУДОВА ПОСЛІДОВНОСТІ ЗОБРАЖЕНЬ ПОВЕРХНІ У ПРОЕКТІ MATHPARTNER

Текстова частина до кваліфікаційної роботи
за спеціальністю 121 «Інженерія програмного забезпечення»

Керівник кваліфікаційної роботи
д.ф.-м.н., проф. Малашонок Г.І.

_____
(*підпис*)
“____”_____ 2024 р.

Виконав студент
Нечипорчук Д.Є.

_____
(*підпис*)
“____”_____ 2024 р.

Київ – 2024

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,
доцент, к.ф.-м.н.
_____ С. С. Гороховський
(підпис)
“____” _____ 20__ р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студенту: 4 р. н. БП ІПЗ факультету інформатики

Нечипорчуку Данилу Євгенійовичу

на тему: Побудова послідовності зображень поверхні у проекті MathPartner

Зміст ТЧ до кваліфікаційної роботи:

Анотація

Вступ

1. Огляд системи MathPartner

2. Аналіз завдання

3. Впровадження рішення для серверної частини

4. Впровадження рішення для користувацької частини

Висновки

Список джерел

Дата видачі “___” _____ 20__ р.

Керівник _____ Завдання отримано _____

Тема: Побудова послідовності зображень поверхні у проекті MathPartner

Календарний план виконання роботи:

№	Назва етапу кваліфікаційної роботи	Термін виконання	Примітка
1.	Отримання завдання на кваліфікаційну роботу	13.11.2023	
2.	Огляд системи MathPartner	11.12.2023	
3.	Огляд літератури за темою роботи	15.01.2024	
4.	Розробка серверної частини роботи	26.02.2024	
5.	Розробка користувацької частини роботи	25.03.2024	
6.	Написання текстової частини роботи	22.04.2024	
7.	Надання роботи керівнику для перевірки	9.05.2024	
8.	Коригування роботи за результатами перевірки	15.05.2024	
9	Подання роботи на кафедру для перевірки на плагіат	20.05.2024	
10.	Захист кваліфікаційної роботи	27.05.2024	

Студент Нечипорчук Д. Є.

Керівник Малашонок Г.І.

“____” _____

Зміст

Анотація 5
Вступ 6
Розділ 1. Огляд системи MathPartner 7
1.1 Створення графіків 7
1.2 Послідовність кадрів 8
1.3 Побудова графіка на користувацькій частині 9
1.4 Обробка запиту серверною частиною 11
Розділ 2. Аналіз завдання 13
2.1 Формулювання завдання 13
2.2 Варіанти взаємодії користувацької частини та серверної 13
2.3 Варіанти відображення кадрів на користувацькій частині 15
Розділ 3. Впровадження рішення для серверної частини 16
3.1 Обробка першого запиту 16
3.2 Обробка запиту для кожного кадру у класі Page 17
3.3 Обрахунок даних про графік для кожного кадру 19
3.4 Залежності для серверної частини 19
3.5 WebSocket конфігурація 20
3.6 Обробка повідомлень, що надіслані через WebSocket 21
Розділ 4. Впровадження рішення для користувацької частини 24
4.1 Залежності для користувацької частини 24
4.2 Відправка та отримання повідомлень 25
4.3 Зміна об’єктів three.js 26
Висновки 28
Список джерел 29

Анотація

У кваліфікаційній роботі було розглянуто способи покращення хмарної платформи MathPartner для побудови 3D графіків з параметрами. Було впроваджено рішення з використанням WebSocket технології для серверної частини застосунку, яка написана мовою Java та використовує фреймворк Spring. Рішення для користувацької частини використовує мову JavaScript та бібліотеку Three.js, щоб забезпечити відображення зміни параметрів графіка у вигляді послідовності зображень.

Ключові слова: MathPartner, графік функції, функція з параметрами, послідовність зображень, Spring, WebSocket, STOMP, Three.js.

Вступ

MathPartner [1] – це комплексна система, яка дозволяє користувачу проводити безліч різноманітних математичних розрахунків. Однією зі складових цієї системи є модулі, які відповідають за створення 2D та 3D графіків функцій. Система надає можливість створити графік функції, який містить параметри. Ці параметри можуть набувати різних значень і, як наслідок, змінювати графік. MathPartner дозволяє вказати діапазон для кожного параметра і показати як змінюється графік при зміні значення параметрів протягом вказаної кількості кадрів.

Наразі система MathPartner пропонує декілька різних способів побудови графіків. Один з них – це побудова зображень 3D графіків на клієнтській частині за допомогою бібліотеки Three.js [2], яка дозволяє будувати будь-які 3D фігури та використовує WebGL [3]. Метою моєї роботи буде покращення цієї частини MathPartner, щоб впровадити можливість перегляду зміни значень параметрів функції у вигляді послідовності кадрів.

Завдання можна визначити наступні: проаналізувати усі наявні способи побудови графіків функцій та послідовності кадрів у системі MathPartner, дослідити способи реалізації зміни побудованих фігур у бібліотеці Three.js, покращити серверну та клієнтську частину MathPartner, щоб забезпечити можливість демонстрації зміни значень параметрів функції шляхом зміни послідовності зображень.

Розділ 1. Огляд системи MathPartner

1.1 Створення графіків

У системі MathPartner реалізовано створення 3D графіків різними способами. Першим з них є створення зображень графіка на серверній частині, прикладом якого є використання методу \plot3d. Для початку користувач має ввести у систему запит, що схожий на наступний:

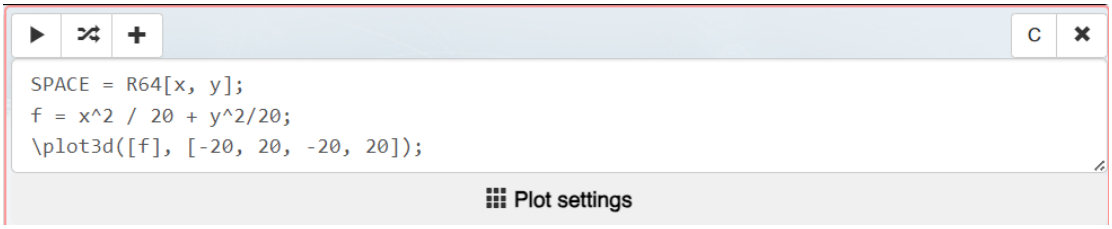

Рис. 1.1 Приклад запиту користувача у системі MathPartner

Запит на рисунку 1.1 складається з трьох частин: оголошення аргументів з використанням ключового слова SPACE, оголошення функції, яка позначена літерою f, та методу \plot3d, який приймає функцію та додаткові параметри. Як додаткові параметри, метод \plot3d отримує мінімальне та максимальне значення для аргументів функції, адже система при побудові буде перебирати з певним кроком всі можливі значення цих аргументів. Методу \plot3d достатньо максимального і мінімального значення для двох аргументів, які зазвичай виглядають як x та y, тому на рисунку 1.1 він приймає 4 числових значення окрім функції. Надалі запит буде відправлений до серверної частини й опрацьований нею.

Спочатку сервер проведе обробку запиту, щоб виділити функцію, аргументи та додаткові параметри. Згодом буде обраховано всі точки, яких набуває функція у вказаному запиті, враховуючи діапазон для x та y. На основі цих точок буде створено зображення графіка, використовуючи одну зі стандартних бібліотек у мові Java - AWT. Як тільки цей етап буде завершено, користувацька частина зможе отримати готове зображення та відобразити його.

Зображення створюється серверною поступово і може займати багато часу. Запит з рисунка 1.1 створить наступне зображення:

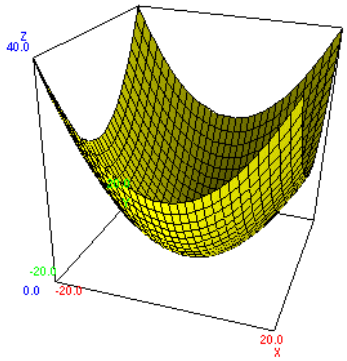

Рис. 1.2 Приклад побудованого на серверній частині 3D графіка у MathPartner

1.2 Послідовність кадрів

Функція може містити не лише два аргументи x та y, їх може бути безліч. Усі аргументи, які не є першими x та y, я буду називати додатковими аргументами або параметрами. Якщо функція містить такі параметри, то на етапі створення зображення їм буде надано значення 1.

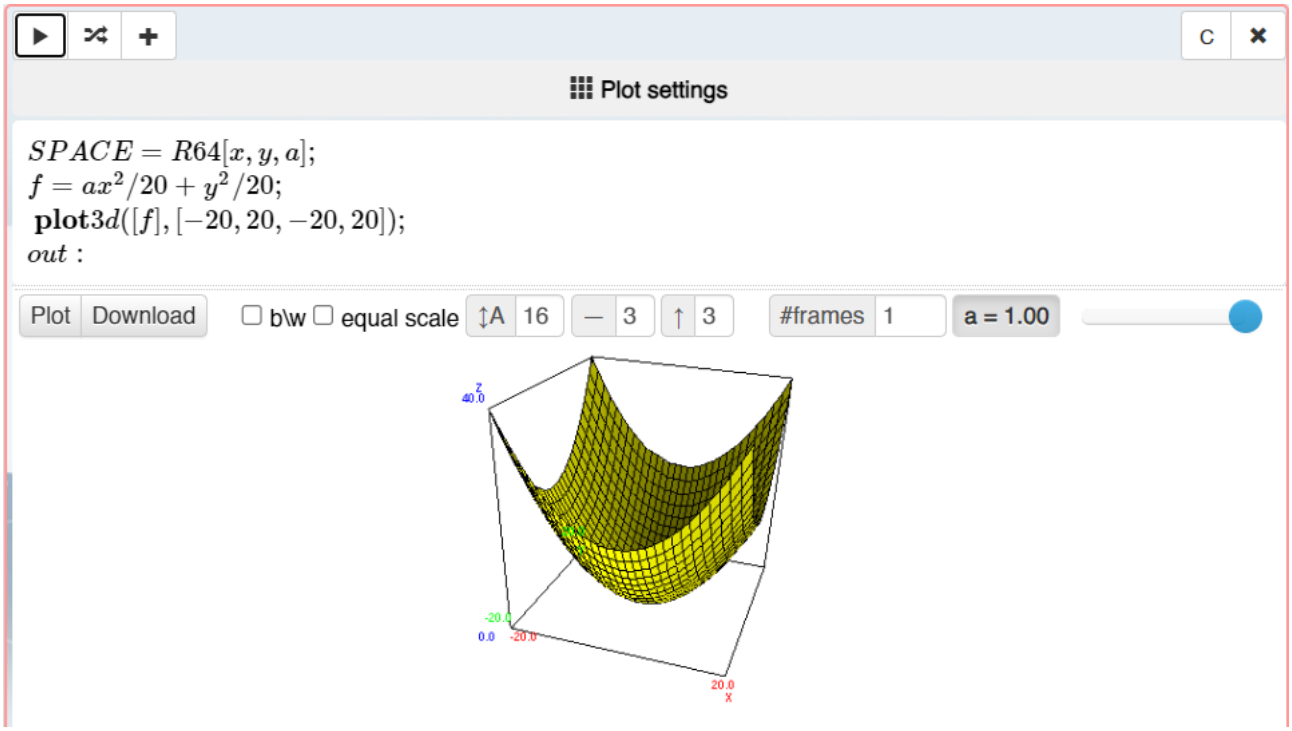

Рис. 1.3 Створення графіка з додатковим аргументом

При множенні першого доданка на параметр “a” у функції на рисунку 1.1, ми отримаємо таке ж саме зображення графіка, бо аргумент “a” буде дорівнювати 1. Такий результат можна побачити на рисунку 1.3. Додаткові аргументи можна використати для створення послідовності зображень, вказавши діапазон для кожного з них.

Після першого відображення графіка, на сторінці з’являться нові кнопки, поля та повзунки, які помітні на рисунку 1.3 і знаходять між текстом запиту та зображенням графіка. Для кожного параметра функції буде відображено окремий повзунок. Оскільки початково всі параметри дорівнюють 1, то обрати на ньому потрібно лише кінцеве значення. У полі #frames потрібно вказати цілим числом кількість кадрів. При виборі значення 0 за допомогою повзунка для параметра “a” і значення 5 для кількості кадрів, ми отримаємо 5 зображень, на яких аргумент “a” буде набувати значення 0.8, 0.6, 0.4, 0.2, 0. Результатом побудови послідовності кадрів будуть наступні зображення:

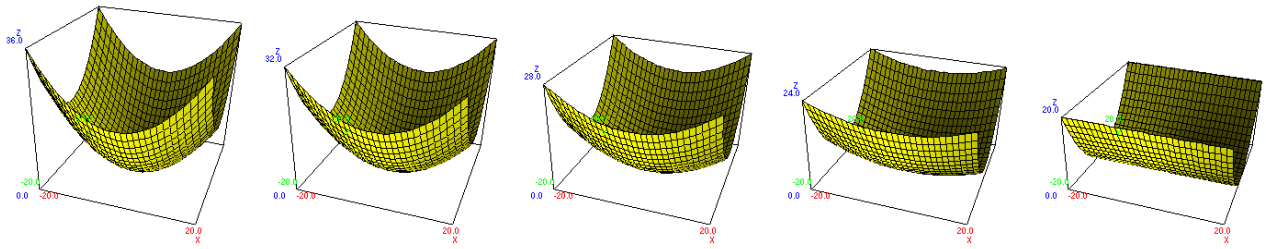

Рис. 1.4 Приклад кадрів при зміні параметра

1.3 Побудова графіка на користувацькій частині

Також MathPartner пропонує альтернативний спосіб побудови 3D графіків - створювати зображення на користувацькій частині з використанням бібліотеки Three.js. Для створення таких зображень потрібно використати інші методи: \explicitPlot3d, \implicitPlot3d та \parametricPlot3d. Метод \explicitPlot3d може побудувати графік функції від двох аргументів, які у просторі є координатами x та y. Третя координата, яка позначається як z, буде дорівнювати значенню функції. Метод \implicitPlot3d приймає функцію одразу від трьох аргументів і будує графік з розрахунком, що ця функція дорівнює

нулю. Метод \parametricPlot3d приймає три функції від двох аргументів, по одній для кожної координати.

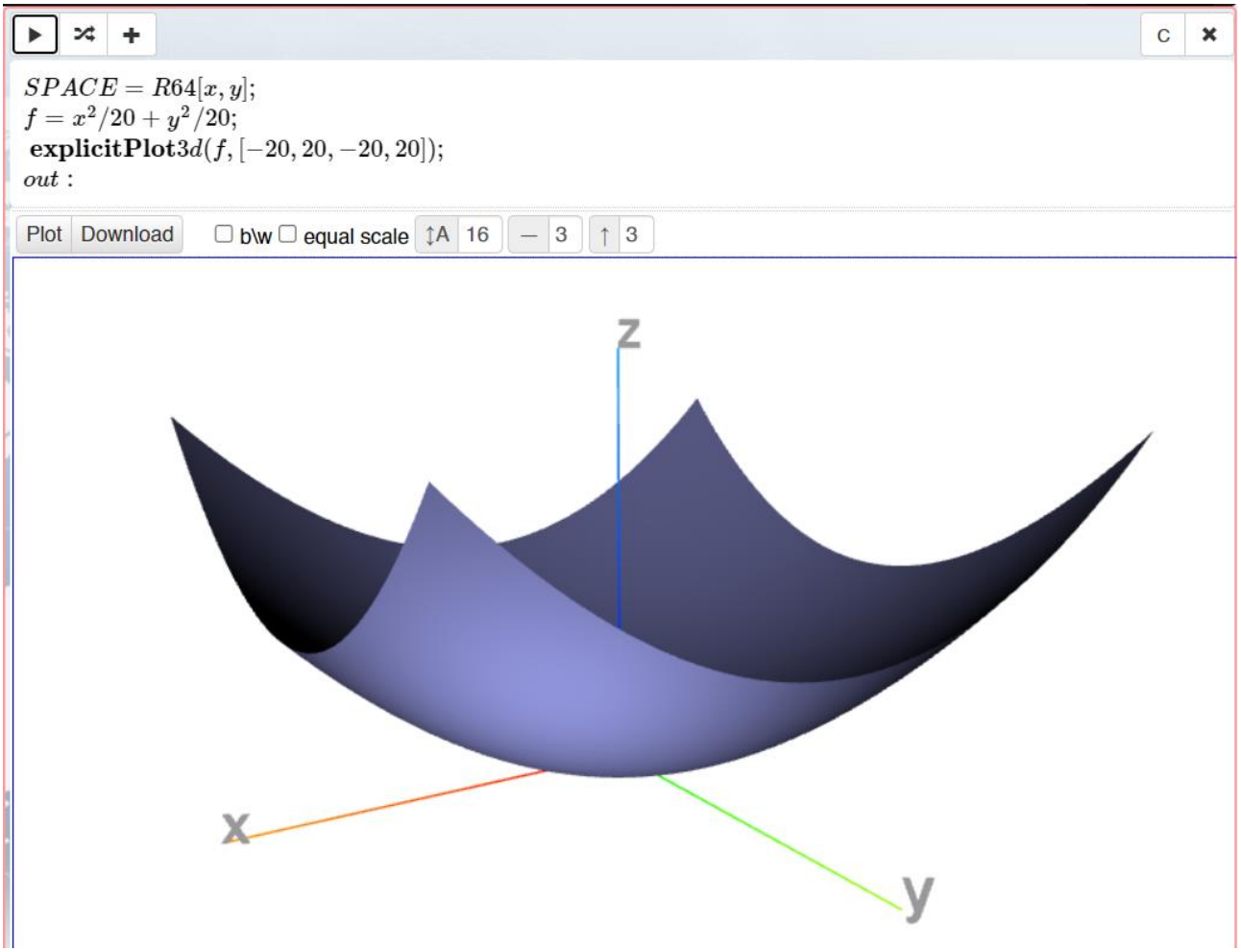

Рис. 1.5 Результат виконання \explicitPlot3d

Перші етапи створення зображення з рисунка 1.5 будуть схожими на минулий метод: серверна частина оброблює запит, виділяє функцію з аргументами, обраховує всі точки, яких набуває ця функція. Точки будуть записані у масив і повернуті до користувацької частини. Своєю чергою користувацька частина має на основі цих точок створити зображення самостійно. Для початку створюється Three.js сцена, яка міститиме усі 3D об’єкти. Далі створюється камера, яка визначає, з якої точки ми дивимося на сцену. Ба більше, камера міститиме такі параметри як кут поля зору, відношення сторін зображення та максимальну й мінімальну відстань для відображення об’єктів.

Після усіх вище зазначених налаштувань можна наповнювати сцену об’єктами. Першими створюються координатні осі x, y та z, які дозволять користувачу легше орієнтуватися у просторі. Згодом до сцени буде додано світло, яке дозволить краще бачити об’єкти та спричинить появу тіней. На останок буде створено об’єкт Geometry, який є основною фігурою. Фактично цей об’єкт геометрії є поверхнею графіка і складається з усіх точок графіка, які ми отримали від серверної частини. Ці точки будуть додані у масив з вершинами, який є полем об'єкта геометрії. Також геометрія вимагає заповнити масив з гранями, у якому зберігається інформація про те, вершини з якими індексами обмежують грані фігури. Перед тим як додати геометрію до сцени необхідно створити об’єкт Mesh на її основі. Mesh, окрім уже описаної геометрії, містить також інформацію про матеріал та колір фігури. Після виклику методу render, ми нарешті побачимо наш графік.

Додатково у сцені налаштовані анімації. Вони дозволяють користувачу рухатися по сцені за допомогою миші: як приклад наближатися чи віддалятися від фігури, що створює ефект збільшення або зменшення масштабу графіка відповідно.

1.4 Обробка запиту серверною частиною

У випадку побудови зображень на користувацькій частині, вона звертається до сервера за посиланнями “/api/plot3dexplicit”, “/api/plot3dimplicit” та подібними. Всі ці посилання описані у файлі Plots.java як окремі методи класу-контролера і всі вони мають схожу структуру. Кожен метод приймає два параметри: об’єкти MathparRequest та Page. MathparRequest – це клас, що містить поле task, для зберігання запиту користувача у вигляді стрічки, та поле sectionId, для роботи з декількома секціями запитів користувача на одній сторінці. Цей об’єкт заповнюється даними від користувацької частини. Екземпляр класу Page створюється і надається завдяки власній анотації Page та класу PageArgumentResolver. Цей клас забезпечує створення об'єкта Page при

першому зверненні до сервера. Надалі цей об’єкт буде зберігатися як атрибут http сесії та використовуватися при наступних зверненнях.

Методи у контролері Plots оброблюють запит користувача за допомогою спеціальних TimeoutRunner класів, які є окремими для кожного методу. Задача цих класів – це встановити ліміт по часу у 45 секунд для одного запиту користувача та обробити можливі помилки при подальшій роботі з запитом. Наступним кроком запит передається Callable класу, який є також окремим для кожного методу. Цей клас у методі run проводить обробку запиту користувача через екземпляр класу Page та отримує від нього об’єкт типу F, що позначає функцію. Ця функція буде проходити процес обрахунку її точок за допомогою окремих класів для кожного методу. Результат, який буде виглядати як масив з точками та іншими параметрами, буде відправлений як відповідь від серверної частини.

Розділ 2. Аналіз завдання

2.1 Формулювання завдання

Розглянувши наявні функціонал системи MathPartner можна зауважити, що їй бракує можливості перегляду послідовності кадрів після створення графіка на користувацькій частині. До того ж система наразі взагалі не може побудувати графік функції, який містить параметри. Отож, моїм першим завданням буде впровадити заміну усіх аргументів функції, які не є обов’язковими x, y та z, на 1 при першій побудові графіка функції. Після цього потрібно розробити механізм, за яким користувацька частина буде звертатися до серверної, для отримання даних про графік функції з урахуванням значення аргументів для кожного кадру. Згодом потрібно з’єднати користувацьку частину та новостворений функціонал серверної частини. Наостанок потрібно розробити рішення, за яким користувацька частина зможе змінювати представлення графіка функції при зміні кадрів засобами Three.js.

2.2 Варіанти взаємодії користувацької частини та серверної

Отже, користувацька частина має значення додаткових аргументів функції та кількість кадрів. Як реалізувати взаємодію з серверною частиною?

Перша ідея полягає у тому, що користувацька частина звертається до серверної один раз, а та обраховує значення одразу для всіх кадрів. Як відповідь користувацька частина отримає масив довжиною у кількість кадрів, де кожен його елемент - це дані для побудови кадру. Це надійне рішення, яке забезпечить мінімальну кількість запитів на сервер. Ми точно будемо впевнені, що користувач отримає усі кадри та побачить їх у правильній послідовності. Швидкість зміни кадрів буде однаковою для кожного запиту. Недоліком буде дуже велика затримка між створенням запиту та отриманням відповіді. Якщо користувач вимагає 100 кадрів, то він повинен чекати поки всі вони будуть обраховані сервером.

Друга ідея – робити окремий запит для кожного кадру. Користувацька частина буде відправляти значення параметрів до серверної частини у циклі. На кожній ітерації вона створить запит та отримає масив з точками графіка, щоб відобразити кадр. Такий підхід розв'яже проблему довгого очікування користувачем відображення кадрів. Проте цей спосіб створить новий виклик – велика кількість запитів може перевантажити сервер і спричинити сповільнення його роботи або взагалі зупинку. У цьому випадку потрібно, як мінімум, розробити рішення для обмеження кількості запитів для одного користувача. Потенційно друге рішення можна розширити виконанням запитів асинхронно, щоб пришвидшити процес завантаження кадрів. Хоча ідея асинхронних запитів тільки погіршить проблему, що стосується збільшення навантаження сервера. До того ж кадри повинні відображатися у правильному порядку, а тому вимагатимуть певної синхронізації після опрацювання сервером.

Фактично перша ідея була реалізована при створенні 2D та 3D графіків на серверній частині, тобто у випадку використання методу \plot3d. Я вважаю, що велика затримка між відправленням запиту та відображенням кадрів у такого рішення занадто негативно впливає на користувацький досвід. Усі відео програвачі використовують механізм завантаження даних під час перегляду, а впроваджене рішення не передбачає цієї можливості. У результаті я вирішив об’єднати ці рішення, та вдосконалити їх протоколом WebSocket. Цей протокол було описано у стандарті RFC 6455 [4]. За цим стандартом користувач та сервер можуть встановити TCP з’єднання та відправляти по ньому будь-яку кількість даних у дві сторони – як від клієнта до сервера, так і навпаки. З цим підходом користувацька частина надсилає один запит до серверної, а та повертає дані про кадри через WebSocket. Дані будуть надходити поступово, що розв'яже проблему довгого очікування на кадри. Водночас з’єднання є швидким та надійним, що якраз допоможе розв'язати проблему великої кількості запитів на сервер.

2.3 Варіанти відображення кадрів на користувацькій частині

У той чи інший спосіб користувацька частина отримає масив з точками графіка і повинна змінити графік в залежності від кадру. Створений графік є об’єктом Mesh від бібліотеки Three.js, і саме над ним потрібно проводити зміни. Mesh створюється на основі об’єкта Geometry, який складається з окремого масиву вершин та масиву граней. Бібліотека дозволяє вносити обмежені зміни до уже створених об’єктів. У об’єкта Geometry можна змінити лише масив вершин, за умови, що кількість вершин залишиться незмінною. Якщо новий кадр вимагає зміни граней або кількості вершин, то об’єкт Geometry доведеться створювати з нуля. Проаналізувавши код обрахунку точок графіка на серверній частині та код створення графіка на користувацькій частині, я дійшов висновку, що методи \explicitPlot3d та \parametricPlot3d повертатимуть однакову кількість точок для різних кадрів, а \implicitPlot3d повертатиме зазвичай різну кількість. Кількість точок для перших двох методів залежить від параметра gridSize, який надається користувачем при першому створенні графіка і не змінюється протягом обрахунку різних кадрів. Грані для цих методів обраховують в залежності від кількості точок, тому вони теж залишаються незмінними зі зміною кадрів. У випадку \implicitPlot3d, використовується дещо складніший підхід, який повертає різні результати при обрахунку кадрів. Отже, для \explicitPlot3d та \parametricPlot3d достатньо буде змінювати масив вершин в об’єкті геометрії, а для \implicitPlot3d доведеться перестворювати об’єкт геометрії.

Розділ 3. Впровадження рішення для серверної частини

3.1 Обробка першого запиту

Для початку потрібно реалізувати заміну всіх додаткових аргументів функції на 1 при першій побудові графіка. Як було сказано раніше, обробка запиту відбувається у класі Page. Кожен запит являє собою стрічку, яка містить перелік аргументів, функцію графіка та метод MathPartner з додатковими налаштуваннями. Основна задача класу Page – це перетворити стрічку з запитом в екземпляр класу F, щоб передати його іншим класам. Після першочергової обробки стрічки, яка визначає метод запиту, відбувається основна логіка обробки за допомогою оператору switch. Цей оператор має окремі інструкції для усіх методів MathPartner. Методи з якими я працюю, а саме \explicitPlot3d, \implicitPlot3d та \parametricPlot3d, знаходяться поруч у визначенні оператора switch і виконуватимуть однаковий код.

Аргументи функції зберігаються в екземплярі класу Ring, який створюється у процесі першочергової обробки стрічки із запитом користувача. Цей екземпляр містить масив екземплярів класу Polynom, який і містить усі аргументи функції. Першочергова обробка також створює екземпляр класу F, який міститиме основну функцію з запиту. Ця функція буде мати усі аргументи у такому вигляді, у якому їх вказав користувач у стрічці з запитом. Сам екземпляр має цікаву структуру, бо складається з масиву екземплярів класу Element. Наповнення цього масиву залежить від методу MathPartner, але зазвичай містить саму функцію як перший елемент масиву та додаткові параметри у вигляді наступних елементів.

Заміна додаткових аргументів функції на одиниці вже реалізована при використанні методу plot3D. Проаналізувавши це рішення, я зміг адаптувати його під власні потреби. Заміна починається зі створення масиву екземплярів класу Element, довжиною такою ж, як і масив аргументів функції. Він міститиме всі аргументи функції у вигляді або змінної як екземпляру класу Polynom, або числа 1 як екземпляру класу NumberZ. Початково потрібно

скопіювати туди ті аргументи, які не будуть змінюватися. Такими аргументами є типові x та y, але у загальному вигляді ми не змінюємо перші два елементи з масиву аргументів функції. Саме їх і потрібно скопіювати у новостворений масив. Додатково метод \implicitPlot3d вимагає копіювання перших трьох елементів з масиву аргументів, адже у його побудові використовуються три аргументи. Одразу після копіювання, інші елементи масиву стають об'єктами NumberZ.ONE. Далі відбувається клонування екземпляра класу F, який описує функцію. Екземпляр містить саму функцію як перший елемент свого масиву екземплярів Elements. Саме у цій функції потрібно змінити деякі аргументи на одиниці. Зміну провести досить просто за допомогою методу value класу F. Цей метод прийме як параметр масив екземплярів класу Element, який ми створили до цього, та екземпляр класу Ring. Метод повертає екземпляр класу Element, тому нам потрібно привести цей об'єкт до класу F за допомогою відповідного конструктора. Додатково варто зазначити, що при використанні методу \parametricPlot3d користувач вказує аж три функції, для змінних x, y та z. Тому у випадку цього методу змінювати потрібно не лише перший елемент масиву в екземплярі класу F, а й другий та третій.

Наостанок відбувається передача нової функції до наступної частини коду. Така передача відбувається за допомогою поля expr у класі Page. Це поле є списком екземплярів класу Element. Нова функція додається у цей список і надалі може бути використана будь-ким, хто має цей екземпляр класу Page.

3.2 Обробка запиту для кожного кадру у класі Page

Під час створення даних про графік на кожен кадр, серверна частина повинна обробити запит від користувача. Технічно, для кожного кадру потрібно спочатку створити функцію, у якої всі додаткові аргументи замінені на їх відповідні значення для цього кадру. Цей процес дуже схожий на заміну аргументів одиницями, який я описав у розділі 3.1. Щоб використати уже написаний код, я виніс його в окремий метод. Цей метод прийматиме функцію,

аргументи якої треба змінити, та масив екземплярів класу Element. Масив містить значення, які будуть підставленні замість додаткових аргументів функції. Цей метод використовується при обробці першого запиту на створення графіка й отримує як останній параметр масив, який заповнений одиницями.

MathPartner місить окремий метод \replot, один з параметрів якого є масивом значень всіх додаткових аргументів функції. Проаналізувавши роботу цього методу разом з методом plot3D я побачив, що важливу роль при створенні послідовності кадрів грає поле plots у класі Page. Це поле є структурою Map, де ключ – це ціле число-ідентифікатор, а значення – це екземпляр класу F. Після обробки першого запиту від користувача, функція, для якої можна створити кадри, додається до plots. Після наступного запиту з методом \replot, система перевіряє чи знаходиться у plots функція, для якої потрібно провести обробку. Згодом відбувається обробка запиту \replot, щоб виділити усі його параметри. Далі відбувається перевірка через умовний оператор, щоб виконати правильну обробку для методів \plot3D, \paramPlot та інших. Спочатку я змінив обробку першого запиту на побудову 3D графіка, щоб додати до поля plots функцію графіка. При чому функція додається перед зміною її аргументів на одиниці, щоб функцію можна було використати при обрахунку даних для кожного кадру. Згодом, у частину з обробкою методу \replot, я додав перевірку на методи \explicitPlot3d, \implicitPlot3d та \parametricPlot3d. У випадку обробки цих команд, буде викликано метод, який замінить усі додаткові аргументи функції на їх відповідні значення для кадру.

За моєю ідеєю, для кожного кадру буде відбуватися обробка методу \replot. Тоді як параметр методу \replot можна передати масив зі значеннями усіх додаткових аргументів функції для цього кадру. Такий підхід дозволить отримати дані аргументів одразу перед їх підстановкою у функцію без написання додаткового коду, адже обробка параметрів методу \replot уже існує.

3.3 Обрахунок даних про графік для кожного кадру

Обробка запиту для кожного кадру повинна закінчитися створенням масиву з точками. Для існуючих методів побудови 3D графіків цей процес відбувається в окремих TimeoutRunner та Callable класах. Я створив власні класи ReplotGeomtry3dTimeoutRunner та ReplotGeometry3dCallable, які мають отримати стрічку з запитом replot від користувача, що містить значення аргументів функції для певного кадру, і повернути масив з точками графіка. Перший клас викликає другий, огортаючи його перевірками на помилки й обмеженням у часі на 45 секунд. Щоб забезпечити створення масиву з правильними даними для побудови графіка, я використав уже існуючі TimeoutRunner класи всередині класу ReplotGeometry3dCallable. Технічно мій клас викликає обробку запиту користувача через клас Page, екземпляр якого він отримує через конструктор, а потім звертається до ExplicitPlot3dTimeoutRunner, ImplicitPlot3dTimeoutRunner або ParametricPlot3dTimeoutRunner залежно від того, масив з точками якого саме графік нам потрібно створити.

3.4 Залежності для серверної частини

Щоб впровадити зв’язок через протокол WebSocket, спочатку потрібно додати обов’язкові залежності до проєкту. Оскільки вся серверна частина використовує фреймворк Spring [5], то доцільно було використати технології, які підтримуються фреймворком або є його частиною. Оглянувши наявні способи реалізації WebSocket зв’язку у Spring, було прийнято рішення використати STOMP [6]. Це протокол, який є надбудовою над протоколом WebSocket, і використовується для відправлення повідомлень у текстовому або двійковому вигляді. Хочу зазначити, що дуже вдалий приклад був представлений на сайті Baeldung [7], і саме він допоміг мені впровадити зв’язок за допомогою цього протоколу. Отже, до проєкту було додано залежності spring-websocket та spring-messaging.

3.5 WebSocket конфігурація

Наступним кроком було створено базовий клас-конфігурацію WebSocketConfig. Цей клас імплементує інтерфейс WebSocketMessageBrokerConfigurer та два методи: registerStompEndpoints й configureMessageBroker. Перший метод містить визначення шляху “/replot” методом addEndpoint, до якого буде звертатися користувацька частина під час встановлення з’єднання та відправлення повідомлень. Другий метод містить визначення шляху для брокера повідомлень “/queue” та префіксу для усіх шляхів застосунку “/api”. Дані від сервера будуть відправлятися у вигляді такого собі потоку протягом обрахунку всіх кадрів, що є більш типовим для брокера queue, тому саме такою і стала його назва.

Сам клас має дві анотації: EnableWebSocketMessageBroker та Configuration. Перша є обов’язковою для правильного створення брокера повідомлень, а друга необхідна лише для того, щоб Spring створив екземпляр цього класу під час запуску сервера. Раніше я завжди зустрічав у класі з анотацією Configuration окремий метод, що містить анотацію Bean. З таким підходом ми маємо можливість самостійно створити клас, у методі анотованому Bean, будь-яким конструктором, з подальшим викликом методів і визначенням полів. Але як виявилося, мати метод анотований Bean не є обов’язковою вимогою для класу з анотацією Configuration. Ця анотація є лише вказівкою для фреймворку Spring, що він має створити екземпляр цього класу-конфігурації.

Особливу увагу потрібно звернути на клас WebSocketHandshakeInterceptor, який був мною створений і доданий під час виклику registerStompEndpoints у класі-конфігурації. Завдання цього класу – це втрутитися у процес встановлення з’єднання між користувацькою частиною і серверною, щоб отримати доступ до http сесії. Як було сказано у розділі 1.4, кожне звернення до сервера вимагає наявності екземпляра класу Page. Цей екземпляр створюється під час першого звернення користувача та надалі зберігається як атрибут http сесії. Оскільки сервер після звернення користувача

через WebSocket також мусить отримати екземпляр класу Page, нам потрібно дістати його через атрибути http сесії. Отже, клас WebSocketHandshakeInterceptor дістає з сесії атрибут “page”, приводить його до типу Page та кладе у структуру Map, що зберігає як значення екземпляри Object за ключем-стрічкою, щоб при подальшій обробці запиту користувача мати доступ до екземпляра класу Page.

3.6 Обробка повідомлень, що надіслані через WebSocket

У окремому контролері було створено метод для обробки запитів від користувацької частини через WebSocket. Користувацька частина має надати у запиті дані для обрахунку послідовності кадрів. Серед цих даних є наступні: кількість кадрів, кінцеві значення додаткових аргументів функції, ідентифікатор секції, у якій виконується функція, та інші параметри. Для комфортної роботи з цими даними було створено окремий java клас з назвою Replot3DRequest. Наступним параметром методу є структура Map, що зберігає атрибути сесії. Цю структуру ми отримаємо, якщо використаємо анотацію Header зі значенням “simpSessionAttributes”. Серед значень цієї структури буде екземпляр класу Page, який ми дістали з http сесії раніше та додали до Map за ключем “page”. Останнім параметром буде унікальний ідентифікатор сесії з’єднання через WebSocket. Його можна отримати, якщо використати анотацію Header зі значенням “simpSessionId”. Цей ідентифікатор знадобиться для того, щоб надсилати дані з серверної частини лише до одного користувача, а не до усіх, хто має відкрите з’єднання через WebSocket. Ба більше, він дозволить відправляти дані про кадри саме до того користувача, який надав параметри для побудови цих кадрів. Технічно, для ідентифікації користувачів можна використати будь-які дані, наприклад ім’я користувача, але для системи MathPartner рішення з ідентифікатором сесії буде задовільним, бо користувачі можуть робити запити без реєстрації.

Спочатку потрібно обрахувати точки графіка для кожного кадру. Для цього я обраховую, як має змінюватися кожен додатковий аргумент функції протягом зміни кадрів. Щоб знайти це число потрібно від одиниці відняти кінцеве значення аргументу та поділити обраховану різницю на кількість кадрів. У окремому списку я зберігав значення аргументів для поточного кадру. Далі у циклі, кількість ітерації якого дорівнює кількості кадрів, потрібно змінити значення аргументів поточного кадру та створити на їх основі стрічку з запитом “\replot”. Цей запит також міститиме усі інші параметри, які ми отримали від користувацької частини. Особливу увагу потрібно було приділити приведенню типу чисел з рухомою крапкою до стрічки. Оскільки дуже великі та дуже малі числа за замовчуванням записуються з використанням експоненційної форми, саме такий вигляд вони матимуть після приведення типів. Такий запис не може обробити клас Page, тому я мав відформатувати виведення чисел, для відображення їх з максимальною кількістю цифр після коми. Числа типу double у мові Java підтримують приблизно 16 знаків після коми. Згодом запит буде передано до екземпляра класу ReplotGeomtry3dTimeoutRunner разом з екземпляром класу Page та ідентифікатором секції. У результаті ми отримаємо масив, який містить усі дані для побудови кадра на користувацькій частині.

Відправленням масиву буде займатися SimpMessagingTemplate. Це окремий клас, який є частиною фреймворку Spring. Його екземпляр можна отримати за допомогою ін’єкції залежностей у будь-якій частині застосунку. Клас містить корисні методи при роботі з протоколом STOMP. Відправлення повідомлень буде здійснюватися методом convertAndSendToUser. Цей метод буде приймати як параметр унікальний ідентифікатор сесії з’єднання через WebSocket, для того, щоб визначити, якому користувачу відправляти дані. Масив з точками графіка буде основним наповненням повідомлення. Також метод прийме шлях, за яким потрібно відправляти повідомлення. У моєму випадку шлях матиме вигляд "/queue/geometry", враховуючи той факт, що брокер queue був оголошений у класі-конфігурації WebSocketConfig. Щоб

отримати повідомлення, користувач повинен буде підписатися на розсилку повідомлень за цим шляхом. Останнім параметром методу для відправлення повідомлень буде заголовок. І хоч заголовок зазвичай не обов’язково створювати власноруч, мені його створити потрібно. Вимога ця описана у документації інтерфейсу SimpMessageSendingOperations, який імплементує SimpMessagingTemplate. При використанні ідентифікатора сесії для розрізнення користувачів, цей ідентифікатор необхідно додати до заголовка повідомлення [8]. Отже, серверна частина відправить через WebSocket з’єднання стільки повідомлень, скільки кадрів потрібно відобразити користувачу.

Розділ 4. Впровадження рішення для користувацької частини

4.1 Залежності для користувацької частини

У проєкті MathPartner використовується бібліотека requirejs [9]. Вона дозволяє спростити роботу зі сторонніми бібліотеками для проєктів мовою JavaScript. Використовуючи її, всі залежності проєкту записують у файлі main.js, включаючи назву та шлях до файлів бібліотеки-залежності. Згодом кожен файл з розширенням .js може стати модулем, якщо використає наступну структуру: define([...], function(...){...}). У цьому прикладі, define отримує список з назвами усіх залежностей, які потрібні модулю, а function прийме ці залежності як параметри й міститиме основне наповнення модуля.

Першою залежністю є STOMP, оскільки саме цей протокол був використаний у серверній частині. Фактично STOMP є лише стандартом, а його конкретних реалізацій існує багато, у вигляді різних бібліотек написаних мовою JavaScript. Деякі бібліотеки вже застаріли, а інші лише починають розвиватися. Я обрав реалізацію STOMP клієнта від Джефа Месніла [10]. Його бібліотека має назву stompjs, використовується багатьма розробниками, має детальну документацію та зустрічається у багатьох навчальних матеріалах на тему WebSocket та STOMP, зокрема у статті на сайті Baeldung [7]. Отже, я додав файл бібліотеки stomp.js до проєкту. Наступним кроком потрібно було записати назву та шлях до файлу цієї бібліотеки у розділ path файлу main.js. Для роботи з бібліотекою потрібно використати глобальну змінну Stomp. Щоб у застосунку був до неї доступ, потрібно було додали рядок “shim: 'stomp': {exports: Stomp}” у файл main.js.

Друга залежність – це SockJS [11]. Питання чи використовувати SockJS у нинішніх проєктах є доволі дискусійним. З 2018 року браузери почали підтримувати протокол WebSocket за замовчуванням, тому користь від SockJS можна отримати лише у випадку застарілого браузера. Для надійності я вирішив все-таки використати цю бібліотеку. Необхідно було додати файл бібліотеки sockjs.js до проєкту і вказати до нього шлях у конфігурації requirejs,

яка знаходиться у файлі main.js. Сама ж бібліотека містить основний клас SockJS, екземпляр якого необхідно створити для подальшої роботи. Щоб забезпечити таку можливість, необхідно було додати залежність як змінну SockJS у визначенні модуля graphics.js. Надалі об'єкт SockJS можна створити через цю змінну.

4.2 Відправка та отримання повідомлень

У файлі graphics.js відбувається обробка та налаштування всіх систем, що забезпечують створення 3D графіків. Оскільки створення послідовності кадрів уже реалізовано для методу \plot3d, можна прослідкувати як відбувається ця побудова на користувацькій частині. Загалом, до кнопки “plot” прив’язують виконання методу _handleBtnReplotNonTableplot. У цьому методі відбувається обробка таких даних як кількість кадрів та значення аргументів функції, шляхом зчитування даних у відповідних полях та повзунках. Після цього надсилається запит до серверної частини з обробленими даними й викликається інший метод, який має обробити його відповідь. Щоб впровадити правильну обробку у випадку використання MathPartner методів \explicitPlot3d, \implicitPlot3d та \parametricPlot3d, я додав перевірку на ці методи після зчитування даних з полів та повзунків. Якщо користувач ввів один з таких MathPartner методів, то буде викликано JavaScript метод plot3dReplot. Цей метод має впровадити спілкування з серверною частиною шляхом використання протоколу WebSocket.

У методі plot3dReplot спочатку відбувається створення об'єкта SockJS, який приймає як параметр шлях, що був визначений у Spring застосунку. В моєму випадку це “/replot”. Наступним кроком необхідно створити об’єкт Client через виклик методу over(…) з об’єкта Stomp, який приймає об'єкт SockJS як параметр. Уся подальша взаємодія буде відбуватися через об’єкт Client.

Тепер можна встановлювати з’єднання з сервером шляхом виклику на об’єкті-клієнті методу connect(…), який не потребує додаткових налаштувань.

Цей метод може прийняти функцію, яка виконається у випадку успішного з’єднання. У цю функцію помістимо виклик на об’єкті-клієнті методу subscribe(…). Підписка за шляхом “/user/queue/geometry” дозволить нам отримати повідомлення від серверної частини, яке міститиме дані для побудови графіка функції. Цей метод також приймає функцію, яке буде виконана після отримання повідомлення від сервера. Ця функція має виконати обробку даних від серверної частини та викликати логіку, яка змінить об’єкти бібліотеки Three.js на основі нових даних. Також після підписки на розсилку повідомлень, потрібно самостійно відправити повідомлення до серверної частини за шляхом “/api/replot”. Це повідомлення має містити усі поля, що відповідають java класу Replot3DRequest, про який я розповідав у розділі 3.6.

4.3 Зміна об’єктів three.js

Функція, яка обробить дані для побудови графіка поточного кадру, буде окремою для кожного з 3-х MathPartner методів. Оскільки кожен метод містить окремий файл з усією логікою побудови графіка, можна додати до цих файлів метод replot.

У випадку методів \explicitPlot3d та \parametricPlot3d буде виконуватися однаковий код. При створенні графіків у файлах plot3d_explicit, plot3d_implicit, plot3d_parametric, усі Three.js об’єкти зберігаються як окремі поля. Тому у методі replot ми зможемо отримати доступ до об’єкта геометрії через виклик this.geometry. Доступ до масиву вершин а об’єкта геометрії можна отримати через поле vertices. Очищувати цей масив не обов’язково, тому можна просто у циклі змінити всі його елементи на нові, враховуючи що кожна вершина має бути об’єктом Three.Vector3. Після цього потрібно вказати, що вершини були змінені. Для цього достатньо змінити значення поля verticesNeedUpdate в об’єкта геометрії на істину, що сигналізуватиме бібліотеці про необхідність переглянути масив вершин на наявність змін.

При обробці методу \implicitPlot3d потрібно спочатку видалити стару геометрію. Для цього потрібно викликати метод dispose на об’єкті геометрії. Далі можна створити новий об’єкт Geometry та записати його у поле this.geometry. Згодом потрібно у циклі заповнити його масив vertices та faces даними на основі відповіді від сервера. Одразу за цим йде виклик методів computeFaceNormals та computeVertexNormals в об’єкта геометрії. Ці методи забезпечать надалі правильну обробку світла для нашої геометрії. Останнім кроком буде зміна значення поля geometry в об’єкта mesh на нашу нову геометрію.

Висновки

У цій роботі було розширено систему MathPartner можливістю створення послідовності зображень поверхні. Тепер система може створити графіки функцій з параметрами, при використанні MathPartner методів \explicitPlot3d, \implicitPlot3d та \parametricPlot3d. Зміну параметрів можна побачити у вигляді зміни зображень графіка, вказавши діапазон значень параметрів та кількість зображень.

Для покращення швидкодії системи при зверненні до сервера, було використано протокол WebSocket. Користувацька частина має встановити з’єднання з серверною через цей протокол, підписатися на розсилку повідомлень та ініціювати початок побудови послідовності зображень. Серверна частина має поступово відправляти дані для побудови графіка по встановленому з’єднанню, як тільки вона обрахує дані для одного зображення.

Щоб користувацька частина також швидше відображала зміну зображень, було використано різні підходи, залежно від MathPartner методу. Метод \implicitPlot3d створює об’єкт геометрії для кожного нового зображення, але перевикористовує такі об’єкти як Mesh та Material. Методи \explicitPlot3d та \parametricPlot3d використовують лише зміну значень у масиві вершин об’єкта геометрії, що забезпечує відсутність накладних витрат на створення нових об’єктів. Таке рішення зумовлено особливостями роботи кожного з методів.

Список джерел

1. Math partner. URL: https://mathpar.ukma.edu.ua/
2. Three.js documentation. *Three.js – JavaScript 3D Library*. URL: https://threejs.org/docs/
3. WebGLRenderer. *Three.js – JavaScript 3D Library*. URL: https://threejs.org/docs/#api/en/renderers/WebGLRenderer
4. Melnikov A., Fette I. RFC 6455: the WebSocket protocol. *IETF Datatracker*. URL: https://datatracker.ietf.org/doc/html/rfc6455
5. Spring framework documentation. *Spring*. URL: https://docs.spring.io/spring-framework/reference/
6. STOMP protocol specification, version 1.2. *STOMP*. URL: https://stomp.github.io/stomp-specification-1.2.html
7. Intro to WebSockets with Spring. *Baeldung*. URL: https://www.baeldung.com/websockets-spring
8. SimpMessageSendingOperations. *Spring docs*. URL: https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/messaging/simp/SimpMessageSendingOperations.html.
9. RequireJS API. *RequireJS*. URL: https://requirejs.org/docs/api.html
10. Mesnil J. STOMP over WebSocket. URL: https://jmesnil.net/stomp-websocket/doc/
11. SockJS-client. *GitHub*. URL: https://github.com/sockjs/sockjs-client