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

Хмарна система управління освітнім процесом для STEM дисциплін

Текстова частина до дипломної роботи

за спеціальністю „Програмна інженерія”

Керівник дипломної роботи

дфмн, проф _____

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2020 р.

Виконав студент

_____

(прізвище та ініціали)

“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к.ф-м.н.

_____ С. С. Гороховський

13 листопада 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на дипломну роботу

студенту 2-го курсу ІПЗ

Ретізнику Олексію Анатолійовичу

на тему:

Хмарна система управління освітнім процесом для STEM дисциплін

Зміст ТЧ до магістерської роботи:

1. Вступ
2. Використання підходів хмарного програмування в освітніх ресурсах
   2.1. Мікросервісна архітектура для навчальних ресурсів
   2.2. Blue/green підходи для розгортання сервісів
   2.3. CI/CD процес для розробки
   2.4. Реалізація на прикладі open source проекту Mathpar Learning
3. Використання платформи Mathpar Learning для шкільної освіти в Україні.
   3.1. Використання Mathpar Calculator
      3.1.1. Створення навчального матеріалу за допомогою LaTeX інструментарію. A. Створення стандартизованих національних підручників. B. Створення підручника учителем.

3.1.2. Створення самостійних і контрольних робіт, що містять повні рішення, як прототипи рішень школяра у хмарному зошиті.

3.1.3. Використання Mathpar Calculator, як робочого зошита для вирішення математичних завдань.

3.1.4. Використання Mathpar Calculator, як засоба автоматичної перевірки правильності рішення, повторного рішення і демонстрації правильного рішення за запитом учня.

3.1.5. Автоматичне збереження дій учня і правильності вирішення завдань в щоденнику, який доступний шкільної адміністрації, освітньої обласної та національної адміністрації.

3.2. Створення бази задач та завдань для використання в освітньому процесі

3.2.1. А. Створення бази стандартизованих національних підручників. В. Створення бази підручників вчителем.

3.2.2. А. Створення бази стандартизованих національних самостійних і контрольних робіт. В. Створення бази самостійних і контрольних робіт учителем.

3.2.3. А. Створення бази стандартизованих національних планів проходження навчальних дисциплін для трьох рівнів складності. В. Створення учителем бази планів проходження навчальної дисципліни для навчальної групи.

3.2.4. Створення бази "щоденників" для всіх учнів. Автоматичне заповнення, збереження і доступність бази "днеиніков".

3.2.5. Перспективи організації національного моніторингу навчального процесу на основі бази "щоденників".

3.3. Інтеграція хмарних обрахунків в освітній процес математики, фізики, хімії

4. Висновки

4.1. Дислокація і посилання на опис документації всіх розроблених сервісів.

4.2. Основні відмінності даного проекту від аналогічних.

4.3.Можливий план дій з розвитку проекту.

5. Список літератури
6. Додатки

Дата видачі 26 жовтня 2019 р. Керівник _____

Завдання отримав _____

Календарний план виконання дипломної роботи

Тема:

Хмарна система управління освітнім процесом для STEM дисциплін

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на дипломну роботу.	26.10.2019	
2.	Огляд технічної літератури за темою роботи.	12.11.2019	
3.	Проектування технічного завдання та технічної документації для веб застосунку.	30.11.2019	
3.	Проектування мікросервісної архітектури для застосунку.	12.12.2019	
4.	Проектування CI/CD процесу для застосунку.	20.12.2019	
5.	Реалізація сервісів застосунку.	15.02.2020	
6.	Реалізація CI/CD процесу для застосунку.	02.03.2020	
7.	Створення інструментів для проведення автоматичного тестування.	28.03.2020	
8.	Створення документації для вихідного коду застосунку.	14.04.2020	
8.	Аналіз отриманих результатів.	01.05.2020	
10.	Корегування роботи за результатами попереднього захисту.	15.05.2020	

11.	Остаточне оформлення пояснювальної роботи та слайдів.	20.05.2020	
12.	Захист магістерської роботи (проекту)	01.06.2020	

Студент _____

Керівник _____

“_____”_____

Зміст

Анотація

Використання підходів мікросервісної архітектури та хмарних технологій (в т.ч. обчислень) для освітніх проектів має довго термінові переваги. В ході даної роботи були розглянуті основні переваги їх використання, проведена та описана робота над реалізацією освітнього проекту, який використовує ці підходи. Створений проект носить назву Mathpar Learning. Окрім того, під час створення проекту були розглянуті варіанти подання навчальних матеріалів у електронному вигляді, імпортована існуюча бібліотека електронних навчальних матеріалів для фізики, хімії та математики. Також була проведена робота над інтеграцією хмарних обчислень в освітній процес на прикладі використання системи mathpartner як зовнішньої залежності проекту.

Вступ

З стрімким розвитком технологій в сучасному світі, все важче стає встигати використовувати нові напрацювання у освітньому процесі. Розвиток хмарних технологій дозволяє сильно спростити і здешевити розробку та підтримку освітніх ресурсів, при коректному використанні. Використання мікросервісної архітектури дозволяє розбити складні навчальні застосунки на окремі сервіси, що спрощує загальну підтримку та розробку застосунку. Використання хмарних провайдерів дозволяє спростити процес забезпечення інфраструктури з використання підходу «Інфраструктура як код» (IaaC), її підтримки. Також використання хмарних провайдерів забезпечує економію коштів при впровадженні пауз в роботі завдяки оплаті лише робочих годин застосунку та можливості автоматичної підтримки необхідних кількостей інфраструктурних елементів залежно від навантаження. Впровадження хмарних обчислень в освітній процес також несе безліч переваг, починаючи з інтерактивних лекційних матеріалів, закінчуючи гнучкими пісочницями для експериментів та апробації власних рішень поза межами шкільних матеріалів.

Саме тому завданням даної роботи є розробити комплекс загальнодоступних матеріалів та ресурсів, які спростять процес адаптації вищезгаданих технологій в освітньому процесі. Задля досягнення цього результату необхідно проаналізувати особливості архітектури хмарних технологій, систематизувати та використати в розробці базові підходи до роботи з мікросервісними застосунками. Також необхідно розробити план інтеграції хмарних обчислень в навчальну систему, описати спосіб роботи з системою та гнучкість в її підтримці та використанні. Останнім, але не менш важливим завданням буде надати рекомендації та практичні приклади використання сучасних підходів до імплементації неперервної інтеграції та неперервної доставки (CI/CD).

В якості матеріалів для аналізу були взяті основні роботи та вебінари проведені командами AWS (Amazon Web Services) та MS Azure (Microsoft azure) як лідерів серед хмарних провайдерів. Окрім того були проаналізовані навчальні матеріали та посібники надані сервісом Mahtpar для використання в якості провайдера хмарних обрахунків. Були опрацьовані та систематизовані навчальні матеріали перенесені в систему для використання в навчальних планах.

По завершенню виконання роботи були розроблені рекомендаційні матеріали по кожному з вищеописаних завдань. Був створений програмний продукт з відкритим кодом, який реалізовує рекомендації та є прикладом готової для використання навчальної системи. Кожен компонент системи має чітку технічну документацію і є покритим автоматизованими тестами. Також розгорнутий демо-проект містить імпортовану базу матеріалів та кілька шаблонів навчальних планів для можливості протестувати основні функції.

Результати роботи можуть бути використані для створення власного навчального проекту, який буде використовувати основні підходи хмарних технологій та інтегровані хмарні обрахунки. Також матеріали роботи можна використовувати для впровадження відповідних технологій в проекти тематика яких відрізняється від навчальної завдяки детальній документації та матеріалам.

Глава 1. Використання підходів хмарного програмування в освітніх ресурсах

1.1. Мікросервісна архітектура для навчальних ресурсів

Мікросервісна архітектура - це спосіб проектування веб-застосунків, при якому вся бізнес модель поділяється на незалежні модулі, кожен з яких відповідає винятково за свою частину бізнес логіки. Мікросервісна модель концентрується на великій кількості маленьких за розміром незалежних сервісів. Завдяки цьому спрощується розробка застосунку загалом, адже кожен модуль має просту структуру завдяки обмеженій кількості функцій. Окрім того знижується зв'язність застосунку, що допомагає уникнути проблем сильнозв'язних монолітів, таких як складність модифікацій та тестування, відсутність гнучкості.

Додаткові переваги мікросервісної архітектури спостерігаються для навчальних проектів. Оскільки структура навчальних закладів є переважно схожою, спільна частина проектів (така як менеджмент користувачів, логіка навчальних груп, класів тощо) може бути винесена в окремі загальнодоступні модулі, перевикористані в багатьох проектах. Такий підхід дозволить скоротити витрати при розробці нових навчальних проектів та сконцентруватись на безпосередньо розробці унікального функціоналу.

Зазвичай використання мікросервісної архітектури супроводжується використанням хмарних провайдерів для досягнення максимально можливої кількості переваг. За замовчанням хмарні провайдери надають можливість автоматизованого процесу створення необхідної інфраструктури для роботи застосунку. Але окрім того зазвичай надаються додаткові інтегровані сервіси та функції для спрощення та підвищення ефективності певних частин застосунку. Прикладами таких сервісів є AWS Batch, AWS Api Gateway тощо. Але їх використання прив'язує застосунок до конкретного хмарного

провайдера. Оскільки в межах роботи будуть надаватись загальні рекомендації та розроблятимуться незалежні від зовнішніх провайдерів приклади, використовуватись сервіси не будуть. Окрім того специфічні для конкретного хмарного провайдера способи розгортання сервісів (наприклад Cloudformation для AWS) теж використані не будуть. Натомість реалізований альтернативний підхід до інфраструктури як коду з використанням скриптів та Docker контейнерів.

Розглянемо основні підходи до проектування мікросервісних застосунків. Провідною ідеєю архітектури є розбиття застосунку на окремі та незалежні сервіси (сервера), які обмінюються інформацією використовуючи протоколи для міжсерверного спілкування. Найбільш популярним, розповсюдженим та рекомендованим протоколом для спілкування є REST завдяки своїй простоті та структурованості. Інший розповсюджений варіант - Messaging Queues, черги повідомлень. Але для їх використання необхідний окремий брокер-сервер, їх налаштування потребує часу, а додаткові обмеження звужують коло застосувань. Тому для міжсерверної комунікації буде рекомендуватись та використовуватись саме REST.

Важливим у розумінні мікросервісної архітектури є підхід “один сервіс - одна відповідальність”. Фактично цей підхід означає що кожен сервіс у системі повинен бути сконцентрованим на єдиній функціональності, за яку він несе відповідальність. Також це означає, що жоден з сервісів не повинен ніяким чином підлаштовуватись під будь-який інший сервіс. Необхідна функціональність надається через документоване API, яке є вхідним пунктом до роботи з мікросервісом. Також це означає що кожен сервіс сам відповідає за валідацію запитів, аутентифікацію та авторизацію, збереження цілісності тощо.

Розробляти кожен сервіс необхідно з розумінням того, що його процес розгортання є так само незалежним. Це накладає додаткові обмеження для

збереження консистентності і уникнення поломок в системі загалом. Фактично це означає, що розробник не може змінювати уже існуючі API ендпоінти оскільки вони можуть використовуватись іншими сервісами і, відповідно, їх зміна призведе до поломок. Для внесення змін в роботу тої чи іншої частини логіки потрібно представляти версійність ендпоінтів. Таким чином, при необхідності змінити логіку застосунку, вводиться нова версія зміненої частини застосунку (наприклад /user/create -> /v2/user/create). В документацію вноситься нова версія ендпоінту, а розробники інших сервісів отримують завдання перейти на нову версію. Оскільки сервіс досі підтримує і нову, і стару версію ендпоінту, залежні сервіси мають достатню кількість часу для модифікації та власних релізних циклів. Коли всі сервіси успішно переходять на нову версію логіки, застаріла може бути видалена.

Іншим важливим для усвідомлення моментом мікросервісної архітектури є факт того, що кожен сервіс має незалежну інфраструктуру та може бути написаний на будь-якій мові програмування. Оскільки кожен сервіс є незалежним, а рекомендований протокол спілкування REST може бути реалізований незалежно від технічних специфікацій сервісу, стек технологій для кожного з них може бути абсолютно будь-яким. Незалежна інфраструктура означає що будь-які необхідні ресурси для сервісу повинні бути надані окремі та незалежні. Так, наприклад, база даних є окремою для кожного сервісу який потребує збереження даних. Це дозволяє уникнути проблем інтеграції різних сервісів, оскільки кожен з них відповідає винятково за свої ресурси, підсилює стійкість до помилок, оскільки похибки одного сервісу не будуть впливати на похибки іншого, та дозволяють окремі розгортання сервісів.

1.2. Blue/green підходи для розгортання сервісів

Blue/green (A/B) підхід до розгортання сервісів полягає в можливості виконувати будь-які операції над застосунком без видимості для користувача.

Іншими словами, при використанні даного підходу процес розгортання нових версій, повернення до старих версій або перезапуск сервісів є безшовним та безперервним. Тобто будь-який користувач який користувався старою попередньою версією застосунку почне використовувати нову версію застосунку без необхідності виконання додаткових дій (перезавантаження сторінки абощо). Окрім того, цей підхід дозволяє виконати швидке відновлення застосунку до попереднього стану (настільки ж невидиме для користувача) в разі помилок нової версії.

Зазвичай для досягнення такого результату необхідно розробляти архітектуру застосунку з певними обмеженнями, щоб такий безшовний перехід був можливий та не порушував цілісність розробленої системи. Наприклад, варто відмовитись від використання мережевих сесій, оскільки A/B якщо сервіс має стан (stateful), його перезапуск без втрати цього самого стану стає набагато важчою та небезпечнішою задачею. Саме тому при розробці застосунку необхідно одразу визначитись з технологіями які будуть використані і проводити розробку з врахуванням їх обмежень.

Обмеженнями B/G підходу для розгортання сервісів, які були враховані під час розробки сервісу є:

- Відсутність станів сервісу;
- Консистентність відкладених завдань;
- Контроль фонових запланованих завдань;
- Версіювання будь-яких змін в видимих назовні частинах функціоналу
- Імплементація логіки відновлення бази даних до попереднього стану (roll-back скрипти)

Розглянемо кожне обмеження та метод його реалізації поокремо:

Відсутність станів сервісу - для реалізації даного обмеження під час архітектури застосунку уникалось будь-яке збереження короткотривалої

інформації в оперативній пам’яті. Якщо необхідність використовувати короктотривалу інформацію все ж виникала (стан користувача), вона зберігалась у базі даних. Це знижує ефективність роботи застосунку порівняно з використанням оперативної пам’яті, але дозволяє нам повністю позбутись станів, що також позитивно впливає на розподілення навантажень (Load-balancing). Адже замість прив’язки користувача до конкретного екземпляру сервісу через збереження його стану в оперативній пам’яті, користувач в будь-який момент часу може бути направлений на будь-який екземпляр сервісу оскільки вони всі працюють з однією й тією ж базою даних і стан користувача (короткотривала інформація) може бути отримана з неї.

Консистентність відкладених завдань - досить поширеним підходом до роботи з складними запитами, які вимагають важких розрахунків є використання відкладених завдань, тобто задач які будуть виконані у фоні, без необхідності користувачу очікувати їх результату в початковому запиті. Це дозволяє покращити досвід використання сервісу користувачу за рахунок розблокування інтерфейсу та взаємодії з сервісом під час виконання складного запиту. Реальну відповідь на свій запит (результат підрахунків) користувач отримує згодом в окремому запиті. Для отримання відповіді можна використовувати веб-сокети або поллінг стратегії. Оскільки веб-сокети створюють стійке з’єднання, що суперечить попередньому підходу до архітектури для B/G розгортання сервісів, цей варіант відкидається. Залишається використання поллінг стратегії. Вона полягає в «опитуванні» серверу для отримання відповіді. Розрізняють 2 види поллінг стратегій – довготривала та короткотривала. Довготривала стратегія полягає в відправці єдиного запита з довгим очікуванням на відповідь. Такий запит затримується на сервері (залишається відкритим, в сплячому режимі) очікуючи формування відповіді. Якщо відповідь готова – користувачу надсилається відповідь. Якщо час очікування перевищений – запит повторюється до успішного отримання

відповіді. Короткострокова стратегія полягає в відправці запитів на сервер з певними інтервалами та отриманні моментальної відповіді. У випадку якщо відповідь не готова повертається моментальне повідомлення, після отримання якого клієнт відправить запит знову через певний інтервал. Якщо відповідь готова – вона повертається і клієнт припиняє надсилати запити. Оскільки затримка відповіді при довгостроковому очікуванні може розглядатись як «стан» серверу, її використання нас теж не влаштовує. Тому рекомендується використовувати короткострокові опитування. Для покращення алгоритму можна внести додаткові оптимізаційні особливості. Наприклад, якщо тривалість обрахунків, або їх прогрес приблизно відомі, можна повертати їх клієнту для налаштування інтервалу запитів на більш оптимальний або для відображення прогресу користувачу.

Контроль фонових запланованих завдань – якщо сервіс містить якісь фонові задачі для виконання, необхідно переконатись що перезапуск сервісу не призведе до дублювання виконуваної задачі та не перерве виконання існуючого завдання. Для цього існує 2 варіанти реалізації. Перший – створити вікно простою, під час якого сервіс може бути успішно перезапущений без додаткових проблем. Це досягається за рахунок попереднього обрахунку тривалостей завдань та складання негнучкого стабільного розкладу. Очевидно що такий підхід є досить хитким, адже будь-яка модифікація логіки роботи завдань може змінити тривалість її виконання, а відповідно і необхідність перестворення розкладу, що зменшує гнучкість сервісу та підвищує складність його підтримки. Другий підхід до реалізації – використання додаткової системи для менеджменту завдань та реалізація можливості їх продовження з місця зупинки. Даний підхід потребує додаткових затрат на етапі планування та реалізації, але дозволяє значно більш гнучко розробляти застосунок, накладаючи обмеження не на весь сервіс вцілому, а на саме фонове завдання. Така додаткова система може реалізовувати контроль з використанням

таблиць для збереження інформації про стан завдань в базі даних. Хоч це й збільшує складність самих завдань, оскільки всі переходи станів потрібно занотовувати зі звертанням до бази даних, також додаткові затрати виникають при запуску та завершенні самого завдання. Але переваги які цей метод надає, включно з можливістю переглядати хід завдань, їх прогрес, модифікувати їх прямо під час роботи тощо, дозволяють нам розробляти застосунок який підтримує B/G розгортання.

Версіювання будь-яких змін в видимих назовні частинах функціоналу – оскільки головною перевагою B/G розгортання є відсутність потреби в будь-яких додаткових діях зі сторони клієнта/користувача (іншого сервісу) після завершення оновлення, зміни функціоналу повинні бути зворотньо-допустимими, тобто кожна ітерація розробки повинна надавати весь без винятку функціонал попередньої версії в додаток до нових змін. Якщо нова ітерація розробки привносить лише нові функції, додаткові обмеження в межах B/G розгортання не створюються. Інша ситуація виникає у випадку, коли необхідно внести зміни до вже існуючого функціоналу. Під час такої ітерації необхідно забезпечити роботу як попередньої версії (до змін) так і інкрементованої версії (після змін). Для цього вводиться поняття версіювання. Нехай зміни вносяться до існуючого ендпоінту “/user/create”. Замість того щоб безпосередньо вносити зміни в контроллер, логіка якого виконується при отриманні запиту, достатньо створити ще один ендпоінт “ /user/v2/create”. Він буде реалізовувати нову логіку, в той час як “/user/create” буде досі виконувати стару логіку для зворотньої підтримки. У випадку коли зміни відбуваються в публічно доступному ендпоінті (наприклад частина публічного API), старі версії можуть залишатись до тих пір, поки вони не порушують консистентність системи. У випадку внутрішніх сервісів, підтримка старих версій може видалятись одразу після переходу решти компонентів до використання нової версії. Видалення необхідне для зниження кількості коду

та необхідної підтримки/документації, а також для більшої чіткості при розробці внутрішніх систем.

Імплементація логіки відновлення бази даних до попереднього стану (roll-back скрипти) – у випадку якщо нова ітерація розробки має неочікувану або некоректну поведінку, за B/G розгортанням ми повинні відновити застосунок до попереднього стану. Зазвичай це зробити досить просто за рахунок того що під час розгортання ми маємо 2 версії застосунку – нову та стару до тих пір поки не верифікуємо працездатність нової версії. Для відновлення роботи достатньо просто перенаправити весь трафік з нової версії застосунку назад на стару (або не робити перенаправлення на нову, якщо помилка була виявлена до того). Інша ситуація відбувається з базою даних. У випадку коли нова ітерація розробки вносить зміни до схеми бази даних, простої зупинки нової версії сервісу недостатньо. Оскільки відновлення стану є винятковим процесом (за замовчанням поломки які потребують відновлення до попередньої версії повинні відсіюватись ще на етапі розробки) його немає потреби автоматизовувати. Тому достатньо розробити ручний спосіб відновлення стану бази даних до попередньої версії. Такі інструменти як flyway дозволяють використовувати вже готову реалізацію відновлення стану бази даних, проте лише в платній преміум версії, що не є підходящим рішенням для проекту з відкритим початковим кодом який може бути використаний для створення інших проектів. Тому натомість буде використаний аналогічний функціонал в ручному режимі. Для кожної ітерації розробки яка вносить зміни в базу даних будуть створені спеціальні скрипти, які будуть приводити схему до попереднього стану (roll_back скрипти). У випадку невдалого розгортання, вони будуть застосовані в ручному режимі і відновлять схему бази даних до попередньої версії.

1.3. CI/CD процес для розробки

CI/CD процес для розробки – (Неперервна інтеграція/неперервна доставка) – це процес постійної інтеграції змін від команд розробників для застосунку та подальша їх доставка до користувача застосунку шляхом використання налагоджених автоматизованих рішень.

Постійна інтеграція (CI – Continuous integration) полягає в щоденному оновленні кодової бази напрацюваннями команд розробників. Такий підхід дозволяє спростити процес розробки завдяки зменшенню кількості конфліктів під час створення фінального пакунку. Також підхід постійної інтеграції дозволяє створити швидший цикл розробки та отримання зворотнього зв'язку за рахунок швидкої інтеграції функціоналу в проект без необхідності визначати способи версіювання, описувати ці самі версії тощо. Також в межах CI процесу визначаються додаткові дії які автоматично відбуваються при спробі інтеграції. Так, наприклад, можна при кожній спробі інтеграції в головну гілку (створенні запиту на вливання, PR) виконувати юніт та інтеграційні тести для перевірки того, що нові зміни будуть працювати коректно в новому пакунку.

Неперервна доставка (CD – Continuous delievery) – це процес автоматизованої доставки готового пакунку застосунку до клієнтів. Зазвичай такий пакунок зберігається в реєстрі пакунків з унікальною назвою та версією. Під час неперервної доставки цей самий пакунок витягається з реєстру та розгортається в першому середовищі. Потрапивши в це середовище застосунок проходить ряд перевірок, автоматизованих або ручних після чого дається зелене світло на доставку в наступне середовище. Таким чином готовий пакунок подорожує від найнижчих середовищ до фінального середовища в якому ним користується безпосередньо клієнт.

Зазвичай обидва ці підходи використовуються сумісно, оскільки вони чудово доповнюють одне одного. В мікросервісній архітектурі завдяки

простоті компонентів і, відповідно, їх обслуговування, дані підходи та їх поєднання показує себе ще більш ефективно.

Використання неперервної інтеграції. Для використання неперервної інтеграції необхідно визначити набір інструкцій (рекомендацій) для розробників та створити автоматизовані засоби для додаткових дій визначених в межах CI процесу. Серед рекомендацій для розробників можна виокремити:

- Розробка нового функціоналу малими ітераціями, кожна з яких може функціонувати незалежно.

При використання такого підходу ми маємо можливість робити часті оновлення головної гілки кодбази в системі контролю версій таким чином переконуючись що фінальна версія функціоналу не викличе конфліктів з паралельно розроблюваними функціями для цього ж сервісу. Окрім того це надасть можливість впевнено оновлювати кодбазу розроблюваного функціоналу змінами інших команд з головної гілки що дозволить використовувати завжди лише найновіші функціональні частини та уникнути технічного боргу в майбутньому. Додатковою перевагою такого підходу буде швидкий цикл отримання зворотнього зв'язку. Адже завдяки швидкому оновленню головної кодової бази навіть найменші частини функціоналу будуть потрапляти до середовищ для тестування і помилки в їх роботі будуть виявлені набагато швидше. Це дозволить уникнути більших проблем у майбутньому.

- Використання автоматизованих юніт тестів при кожному запиті на оновлення головної кодової бази.

Для того аби переконатись що під час інтеграції ми не отримаємо неробочий застосунок нам необхідно ввести автоматизоване тестування яке відбувається під час кожного запиту на вливання змін. Важливо зауважити, що автоматизовані юніт тести не гарантують нам повністю робочу логіку всього сервісу, а тим паче застосунку. Вони необхідні в першу чергу для того аби

переконатись що проект коректно запускається і кожна його частина (функція) працює як того очікував розробник. Оскільки в якості системи контролю версій проекту буде використовуватись github, для впровадження автоматизованих CI перевірок буде використовуватись сервіс github actions. Цей сервіс дозволяє нам описати необхідні кроки для виконання певних функцій та прив’язати їх до конкретних дій в репозиторії. Таким чином можна описати необхідні кроки для запуску автоматизованих тестів і прив’язати їх до відкриття PR в головну гілку репозиторія. Опис необхідних кроків відбувається у файлах які підпадають формату (з кореня репозиторія)
.github/workflows/[SET_NAME.yml or SET_NAME.yaml],
де SET_NAME – це назва набору інструкцій. Використовуючи цей набір github визначить необхідну інфраструктуру для виконання дій і прив'язку до дій в репозиторії. Щойно будь-яка з дій до яких прив’язаний набір інструкцій буде виконана, github автоматично розгорне необхідну інфраструктуру і виконає всі необхідні функції.

Використання неперервної доставки. Для використання неперервної доставки необхідно визначити список середовищ на які буде відбуватись доставка нових артефактів, спосіб створення нових релізів сервісу та безпосередньо алгоритм їх доставки.

- Список середовищ для неперервної доставки. Ми будемо розглядати 3 середовища в життєвому циклі застосунку. Середовище для розробки (Development) – це середовище в якому розробники інтегрують свої зміни та мають можливість проглянути результати, провести інтеграційне тестування або перевірити необхідні тезиси. Середовище для розробки може бути інколи нестабільним за рахунок процесу розробки. В межах CD зміни на це середовище будуть автоматично потрапляти одразу після вливання в головну гілку репозиторія. QA середовище – це середовище на якому відбуваються всі необхідні

валідації та тести коректності роботи застосунку. Дане середовище повинно мати завжди стабільну версію застосунку з повністю робочою логікою. Застосунок потрапляє на це середовище в момент коли розробники завершують ітерацію розробки для перевірки коректності виконаної роботи та наявності регресії. Production середовище направлене на клієнтів, повинно бути завжди робочим та доступним. Застосунок потрапляє на це середовище після того як пройшло повну верифікацію на QA середовищі.

- Розповсюдження релізних пакунків. Оскільки застосунок використовує Docker контейнери для роботи в середовищі, при кожній доставці достатньо витягти зображення для побудови контейнеру з єдиного реєстру. Оскільки одне й те ж зображення завжди породжує однакові контейнери ми можемо не хвилюватись про втрату конфігурацій або інфраструктурних залежностей під час розповсюдження сервісу від одного середовища до іншого. Для автоматизації цього процесу достатньо створити Jenkins pipeline який буде виконувати необхідні інструкції.
- Створення пакунків. Під час створення пакунку необхідно переконатись що застосунок є в робочому стані, тобто провести всі можливі автоматизовані тести. Якщо тести проходять успішно необхідно запустити процес створення пакунку. Під час цього процесу буде створений виконуваний артефакт готовий до запуску з останньою версією коду. Після цього створюється необхідне docker зображення та вивантажується в реєстр. Якщо все проходить успішно, версія застосунку фіксується, створюється тег системи контролю версій за відповідну версію і версія оновлюється до наступної ітерації.

1.4. Реалізація на прикладі open source проекту Mathpar Learning

Mathpar learning – це навчальний проект з відкритим вихідним кодом, який реалізовує всі підходи описані раніше в роботі а також використовує зовнішні ресурси для виконання хмарних обрахунків про що буде описано згодом.

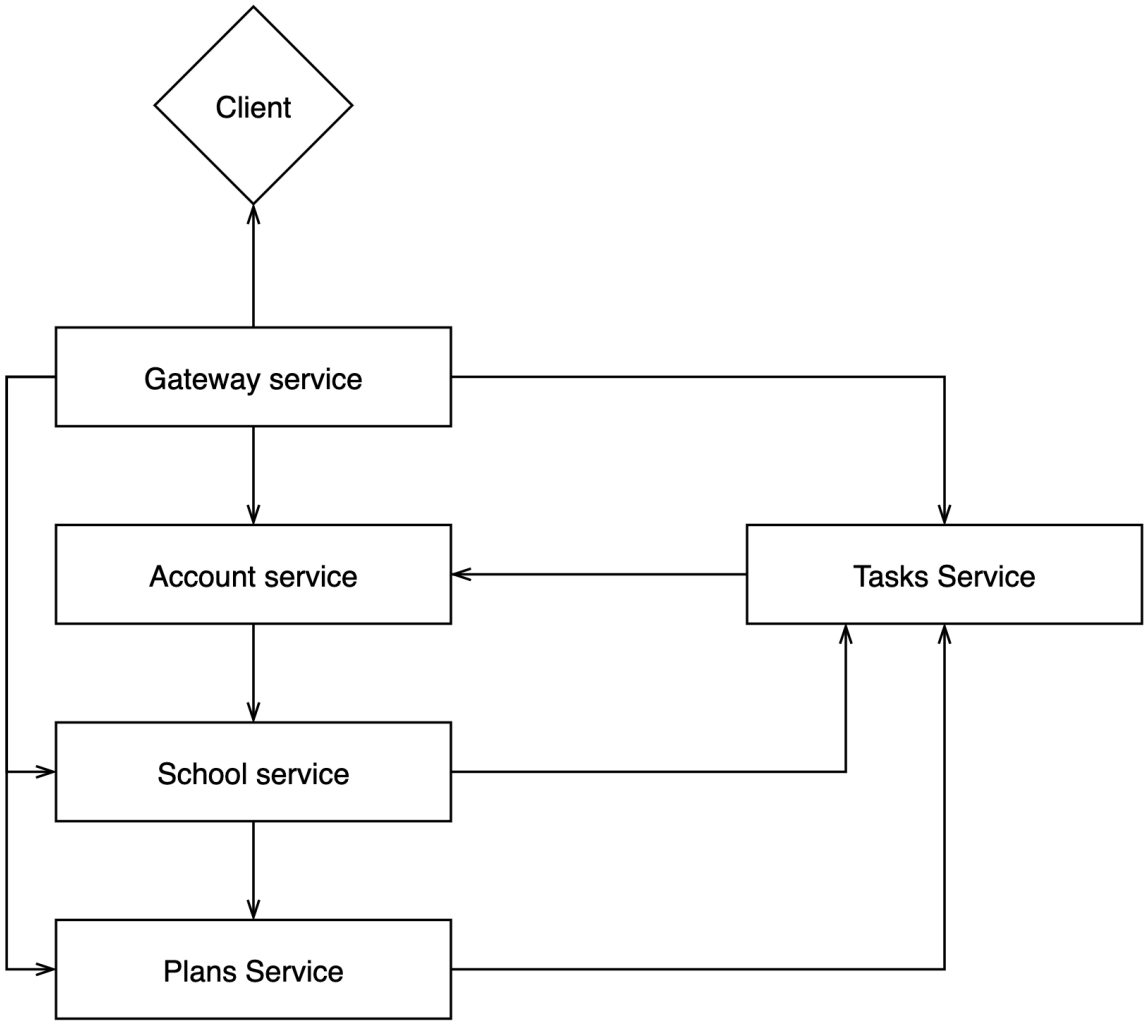
Розглянемо головні ідеї проекту Mathpar Learning для вчителів:

- Можливість вести електронний облік студентів
- Створювати навчальні групи та перерозподіляти студентів між ними
- Створювати навчальні плани для груп
- Проводити дистанційні практичні та контрольні роботи
- Отримувати швидку компактну викладку успішності учнів

Тепер розглянемо головні ідеї проекту Mathpar Learning для студентів:

- Можливість бути прив’язаним до кількох навчальних закладів (збереження історії при зміни шкіл, ВНЗів тощо)
- Можливість проглядати пройдений інтерактивний лекційний матеріал в будь-який момент часу
- Можливість тренуватись в практичних завданнях наданих вчителем необхідну кількість разів
- Можливість проходити екзаменаційні роботи у відведений час
- Можливість проглядати свої успіхи

Для досягнення необхідних ідей притримуючись мікросервісної архітектури застосунку були створені наступні сервіси:



Розглянемо кожен з сервісів поокремо:

Client – це модуль який відповідає за роботу клієнтів (користувачів) з застосунком. У випадку веб-застосунку клієнтом виступає браузерний застосунок. В якості клієнта для RESTful застосунку проект Mathpar Learning використовує фреймворк VueJS. Даний модуль має окремий репозиторій і розробляється повністю незалежно від бекенду. Сам фреймворк дозволяє спакувати усі напрацювання в набір html, js та css файлів для можливості їх використання в браузері.

Вихідний код даного модуля доступний за покликанням: https://github.com/MathparLearningTeam/Client

Gateway Service – це модуль який відповідає за роздачу статичних ресурсів та клієнтського застосунку. Головною його метою є централізувати місцезнаходження ресурсних файлів для можливості їх відокремленого масштабування, захисту та шифрування у випадку необхідності. У випадку зміни архітектурного підходу клієнтського застосунку на багатокомпонентний саме Gateway service буде займатись роздачею кожного з компонентів. Даний модуль може бути реалізований з використанням будь-якої програмної мови або навіть менеджера навантажень. Для простоти та консистентності в межах проекту Mathpar Learning він буде використаний з використанням Java (Spring Framework).

Вихідний код даного модуля доступний за покликанням:
https://github.com/MathparLearningTeam/Gateway

Account service – це модуль який відповідає за роботу з аккаунтами користувачів. Кожен користувач сервісу повинен створити собі аккаунт, який в майбутньому буде його ідентифікаційною картою для різних активностей. Серед активностей можуть бути шкільні (ведення занять як вчитель, адміністрування школи, проходження завдань як студент), так і сервісні (створення та редагування шаблонів завдань, бази підручників, розробка та модифікація планів занять тощо). Також сервіс аккаунтів відповідає за будь-які модифікації аккаунту, такі як відновлення паролю, зміна інформації та безпосередньо видалення аккаунту. У випадку видалення аккаунту на його місці (під його ID) залишається порожній аккаунт щоб уникнути випадкового створення аккаунту новому користувачу з залежностями в інших сервісах. Тобто фактично видалення аккаунту означає лише видалення приватної (PII) інформації.

Вихідний код даного модуля доступний за покликанням:
https://github.com/MathparLearningTeam/Account

School service – це модуль який відповідає за логіку структурних елементів школи. Серед його функцій є: менеджмент користувачів (шкільних профілів аккаунта), авторизація користувачів (визначення ролей), менеджмент класів студентів, менджмент груп студентів, прив'язка навчальних планів (які керуються модулем навчальних планів) до груп, збереження прогресу навчання студентів. Шкільний сервіс не працює напряму з аккаунтами натомість створюючи профілі користувачів. Ці профілі слугують зв'язком «Багато до одного» між школами та аккаунтами відповідно. Профіль також містить інформацію про роль користувача та може містити іншу специфічну до школи інформацію. Профілі обов'язково мусять мати як прив'язану школу, так і аккаунт. Для того щоб створити профіль директора, необхідно зайти в особистий аккаунт і заповнити відповідну форму (школу може створити будь-хто і аккаунт стане директором школи). Коли школа вже існує, її директор або завуч може створити профілі іншим користувачам. Директор, профіль якого створюється автоматично при створенні школи, може створити завучів, вчителів та студентів. Завуч же може створювати лише вчителів та студентів. Вчитель та студент не мають можливості створювати шкільних профілів. Класи можуть бути створені та наповнені/модифіковані лише завучем. Групи ж створюються та наповнюються вчителем. В групу можна додати одразу всіх студентів конкретного класу. Студенти можуть переглядати асоційовані з ними групи (попередньо визначені вчителем) і прикріплені до групи навчальні плани. Кожен студент може проходити завдання та переглядати лекції доступних йому навчальних планів. Весь прогрес студента зберігається і прив'язується до його профілю. Вчителі ж можуть дивитись прогрес кожного студента по навчальному плану в групах доступних вчителю.

Вихідний код даного модуля доступний за покликанням:
https://github.com/MathparLearningTeam/School

Tasks service – це модуль який відповідає за керування навчальними завданнями доступними в проекті Mathpar Learning. Завдання в системі бувають двох типів: «Потребують відповідь» та «Не потребують відповіді». До перших відносяться будь-які задачі з умовою та запитанням для вирішення. Такі задачі в основному використовуються для практичних та екзаменаційних блоків у шкільних планах. Завдання які не потребують відповіді – це радше лекційні матеріали, які створені в першу чергу для сприйняття. Вони досі знаходяться в категорії завдань, оскільки є інтерактивними. Це означає, що кожна лекція написана з використанням розмітки для хмарних обчислень, що дозволяє будь-якому студенту змінити будь-яку частину лекції та переобрахувати результат. Таким чином досягається наступний рівень сприйняття, адже лекції перестають бути просто набором тексту, а стають задачами, які студент може без проблем відтворити та повторити самостійно. Кожна задача може бути створена з нуля, або створена на базі іншої (батьківської) задачі. Ієрархія задач завжди зберігається в базі даних для можливості відтворити оригінал. Авторство будь-якого завдання прив'язується до аккаунта користувача який створив задачу. Оскільки задачі можуть бути використані не лише в шкільних системах, прив'язка йде саме до аккаунта, а не до шкільного профілю.

Вихідний код даного модуля доступний за покликанням:
https://github.com/MathparLearningTeam/Tasks

Plans service – це модуль який відповідає за навчальні плани доступні в проекті Mathpar Learning. Плани бувають двох типів – шаблони та власне робочі плани. Шаблони не можуть використовуватись в якості навчального плану для конкретної групи. Натомість, коли вчитель хоче використати якийсь шаблон для використання в групі, автоматично буде створена копія навчального плану та збережена в базі даних робочих планів. Таким чином після прив'язки плану до групи вчитель має повний контроль над планом і має

можливість змінити будь-які його елементи без модифікацій шаблону який може редагуватись лише автором шаблону. Шаблони зазвичай створюються спеціально навченими людьми або державними установами по контролю якості освіти та відображають рекомендований навчальний процес. У шаблонів можуть бути присутні «мітки» які можна використати для швидкої фільтрації та пошуку між шаблонами. Так, наприклад, державні навчальні плани можуть мати мітку «Державні», навчальні плани для 7х класів можуть містити мітку «7 клас» тощо.

Вихідний код даного модуля доступний за покликанням:

https://github.com/MathparLearningTeam/Plans

Додатково варто виділити **Secrets Manager** який по факту є окремим модулем який відповідає за збереження та контроль вразливих параметрів бекенд сервісів. Такими параметрами є паролі, рядки підключення до бази даних, доступи до поштових сервісів тощо. Його впровадження було вмотивоване в першу чергу cloud-native підходом. Оскільки більшість хмарних провайдерів надають можливість безпечно зберігати вразливі параметри в хмарних сховищах, такий підхід дозволяє полегшити інтеграцію таких сховищ в застосунок. Це відбувається за рахунок того, що кожен бекенд сервіс за замовчанням повинен отримувати вразливі параметри з якогось зовнішнього джерела (в даному випадку – secret manager модуля). Таким чином, при інтеграції хмарного провайдера достатньо лише змінити Spring Bean який відповідає за завантаження параметрів на необхідний для отримання параметрів у хмарного провайдера. Повертаючись до secret manager модуля, - він використовує файлову систему для збереження параметрів. Вони можуть бути закодовані, або в чистому вигляді (перший варіант рекомендований). Файли з параметрами можуть бути змінені в реальному часі без необхідності перезапускати модуль для їх підхоплення. Для зручності менеджменту введені поняття namespace-ів. Кожен неймспейс є скупченням параметрів об’єднаних

за певною характеристикою. Так, наприклад account.namespace – це скупчення параметрів як використовуються account service-ом.

Вихідний код даного модуля доступний за покликанням:

https://github.com/MathparLearningTeam/SecretManager

Проект Mathpar Learning також реалізовує CI/CD підходи в своїй розробці. При кожній спробі зливання змін у головну (integration) гілку запускаються автоматичні github workflow-s (Actions) які проводять юніт тестування. Вихідний код цих дій визначений окремо для кожного репозиторія і знаходиться за шляхом (починаючи з кореня репозиторія) .github/workflows/*.yml. Таким чином перевіряється коректність інтегрованого коду як описувалось у розділі 1.3.

Для реалізації CD процесу використовується Jenkins. Jenkins – це загальнодоступний проект з відкритим вихідним кодом, який дозволяє визначати списки інструкцій для автоматичного виконання та список дій які будуть спричиняти запуск дій. Jenkins має кілька видів задач які він може виконувати, серед них «Вільний проект», «Пайплайн», «Багатогілковий пайплайн». Додаткові види задач можуть бути встановлені з використанням плагінів, але при виконанні проекту Mathpar Learning в цьому необхідності не було. Тому були використані лише «Пайплайни». Для можливості легше та комфортніше організувати роботу а також швидко розгортати нові екземпляри дженкінсу за потреби всі пайплайни створювались у вигляді .Jenkinsfile файлів з специфічним для дженкінсу синтаксисом та зберігались у репозиторії. Створені пайплайни мають декларативний характер, але деякі з них можуть містити скриптовані елементи (declarative vs scripted pipelines).

Весь вихідний код пайплайнів доступний за покликанням:

https://github.com/MathparLearningTeam/Jenkins

Головними пайплайнами розробленими для Mathpar Learning є:

- BuildClient.Jenkinsfile – цей пайплайн відповідає за пакування клієнтського застосунку. Перед пакуванням запускаються юніт тести для валідації працездатності. В результаті виконання створюється jar файл який містить ресурси з клієнтським застосунком. Цей jar файл відправляється в github реєстр для можливості його подальшого використання в Gateway сервісі.
    - Не потребує вхідних параметрів (може потребувати вхідні параметри у випадку переходу на багатомодульний клієнт)

```
  stage('Prepare') {
    steps{
        git url: 'https://github.com/MathparLearningTeam/Client', branch:
'master', credentialsId: 'github-token'
        sh 'git config --global user.email "mathpar.mailer@gmail.com"'
        sh 'git config --global user.name "Mathpar Jenkins"'
        injectCredentials script:this, github: true
    }
  }
  stage('Build artifact'){
    steps{
        sh 'mvn -B build-helper:parse-version release:prepare
release:perform'
    }
  }
```

- BuildService.Jenkinsfile – цей пайплайн відповідає за створення пакунку одного з бекенд модулів. Передусім виконується юніт тестування для перевірки працездатності сервісу. Якщо тести проходять успішно, наступним кроком виконується створення виконавчих файлів (оскільки для бекенд сервісів використовується java – виконавчим файлом є war файли). Під час створення виконавчого файлу відбувається модифікація версії до наступної ітерації розробки. Коли виконавчий файл присутній відбувається відправка його в github реєстр, створення Docker зображення та відправка останнього у реєстр для подальшого використання під час розгортання сервісів. Якщо зображення успішно відправлене у реєстр, задача вважається виконаною успішно.
    - Вхідні параметри: application (вибрати сервіс зі списку)

```
stage('Checkout') {
    steps{
        git url: applications.getSourceUrl(params.application), branch:
'master', credentialsId: 'github-token'
    }
}
stage('Build artifact'){
    environment {
        GIT_SSH_COMMAND="ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no"
    }
    steps{
        sh 'git config --global user.email "mathpar.mailer@gmail.com"'
        sh 'git config --global user.name "Mathpar Jenkins"'
        injectCredentials script:this, github: true, gitlab: true
        sh 'mvn -B build-helper:parse-version release:prepare
release:perform'
    }
}
```

- DeployApplication.Jenkinsfile – цей пайплайн відповідає за розгортання бекенд сервісу в конкретному середовищі. Він повинен запускатись після того як сам сервіс був успішно створений та збережений у реєстрі. Його завданням є зібрати усі необхідні параметри та налаштування для конкретного сервісу воєдино (для досягнення цієї мети використовується IaC репозиторій), сформувати з них команди для запуску та виконати ці команди у відповідному середовищі. IaC репозиторій містить bash скрипти, які в результаті виконання створять та повернуть команди для запуску застосунку. Ці скрипти використовують файли параметрів для модифікацій під конкретне середовище.
    - Вхідні параметри: application (вибрати сервіс зі списку), environment (вибрати середовище для розгортання зі списку), tag (вказати тег зображення сервісу для використання)

```
stage('Deploy'){
    steps{
        //Compose command to run on server using prepared properties
        git url: "https://github.com/MathparLearningTeam/IaC", branch:
'master', credentialsId: 'github-token'
        sshagent(credentials:["ssh-key-
${environments.getShortName(params.environment)}"]) {
            sh script: "cd
${applications.getDeploymentScriptPath(params.application)} &&
COMMAND=\$(bash ./script.sh
${applications.getDeploymentArguments(params.application, params.environment,
params.tag)}) && ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no ${environments.getHostString(params.environment)}
\"\$COMMAND\""
        }
    }
}
```

- DeployInfrastructure.Jenkinsfile – аналогічно до попереднього, мета цього пайплайну є розгорнути інфраструктуру необхідну для роботи застосунку в певному середовищі. Для формування рядка запуску використовується IaC репозиторій, деталі роботи якого описані в попередньому пайплайні. На відміну від попереднього пайплайну, цей не потребує завчасно підготовлених зображень, оскільки використовує публічно доступні зображення інфраструктурних компонентів, конфігурація яких відбувається за рахунок змінних середовища. Тег публічно доступних зображень вказується в файлах параметрів для середовища. Важливо розуміти що даний пайплайн відповідає за будь-які інфраструктурні елементи, а не лише за бази даних. Завдяки гнучкості скриптів у репозиторії IaC він може розгортати будь-якого роду інфраструктуру однією й тією ж послідовністю команд.
    - Вхідні параметри: infrastructure (вибрати інфраструктуру зі списку), environment (вибрати середовище для розгортання зі списку)

```
stage('Deploy'){
    steps{
        git url: "https://github.com/MathparLearningTeam/IaC", branch:
'master', credentialsId: 'github-token'
        sshagent(credentials:["ssh-key-
${environments.getShortName(params.environment)}"]){
            sh "cd
${infrastructures.getDeploymentScriptPath(params.infrastructure)} &&
COMMAND=\$(bash ./script.sh
${infrastructures.getDeploymentScriptPath(params.infrastructure)}
${infrastructures.getScriptArguments(this, params.infrastructure,
params.environment)}) && ssh -o UserKnownHostsFile=/dev/null -o
StrictHostKeyChecking=no ${environments.getHostString(params.environment)}
\"\$COMMAND\""
        }
    }
}
```

Важливо зауважити що пайплайни BuildClient та BuildService виконуються на окремих, попередньо підготовлених агентах (docker контейнерах), які містять всі необхідні залежності для створення та деплойменту необхідних застосунків. Так, для фронтенд сервісу необхідно мати встановлений npm та nodejs, в той час як бекенд сервіси потребують maven та JDK. Додатково варто зауважити, що агенти містять винятково встановлені залежності, жодних параметрів специфічних для проекту, сервісу або середовища вони не містять. Таким чином одні й ті ж агенти можуть бути використані для будь-яких імплементацій. Усі параметри необхідні для роботи пайплайнів зберігаються в захищеному середовищі Jenkins-a і доступаються з використанням ID.

```
agent {
    docker {
        image 'oleksiiretiznyk/maven-npm:latest'
        args '-u 0 -v $HOME/.m2:/root/.m2'
    }
}
```

Окрім того, для спрощення роботи пайплайнів була визначена бібліотека з глобальними змінними та вспоміжними методами. Будь-який Jenkins пайплайн може підключити її з використанням команди “`library 'utils'`”, де `'utils'` – назва бібліотеки в налаштуваннях дженкінса.

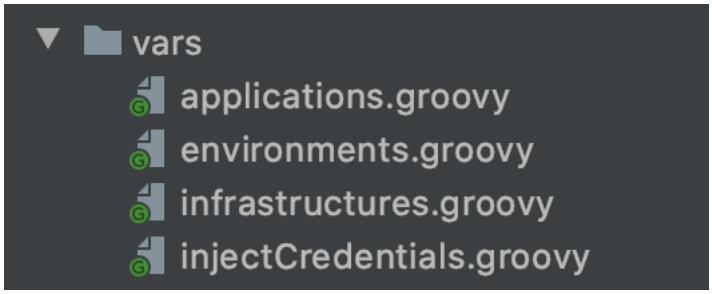


Рис 1.4.1. Вміст бібліотеки ‘utils’

Розглянемо інфраструктурні елементи необхідні для коректного функціонування бекенд застосунків:

- AccountDB – база даних для Account service. В якості реалізації використовується актуальна (lts) версія mysql бази даних. В даній базі даних зберігаються уся інформація щодо аккаунтів, токенів та інша інформація відносно аккаунтів користувачів.
- SchoolDB – база даних для School service. В якості реалізації використовується актуальна (lts) версія mysql бази даних. В даній базі даних зберігається уся інформація щодо шкіл, ієрархії профілів користувачів, їх ролі, успішність студентів, інформація про класи та групи студентів. Також в даній базі зберігають зв’язки з сервісом навчальних планів (прив’язані до груп плани)
- TasksDB – база даних для Tasks service. В якості реалізації використовується актуальна (lts) версія mysql бази даних. В даній базі даних зберігається уся інформація щодо завдань з відповіддю та без відповіді, їхня ієрархія (батьківські зв’язки між завданнями). Також в цій базі зберігається авторство задач (аккаунти які їх створили).
- PlansDB – база даних для Plans service. В якості реалізації використовується актуальна (lts) версія mysql бази даних. В даній базі даних зберігається уся інформація щодо навчальних та шаблонних планів, прив’язаних до них задач та авторство шаблонів (аккаунти які їх створили).

- Поштовий провайдер – в якості поштового провайдеру використовується публічний gmail. Він використовується в якості SMTP серверу для відправки листів. Окремої власної інфраструктури для його використання не потрібно.

Правила розгортання для кожого з цих інфраструктурних фрагментів знаходиться в IaC репозиторії. Там формується рядок запуску застосунку відповідно до параметрів середовища з одного з сумісних файлів.

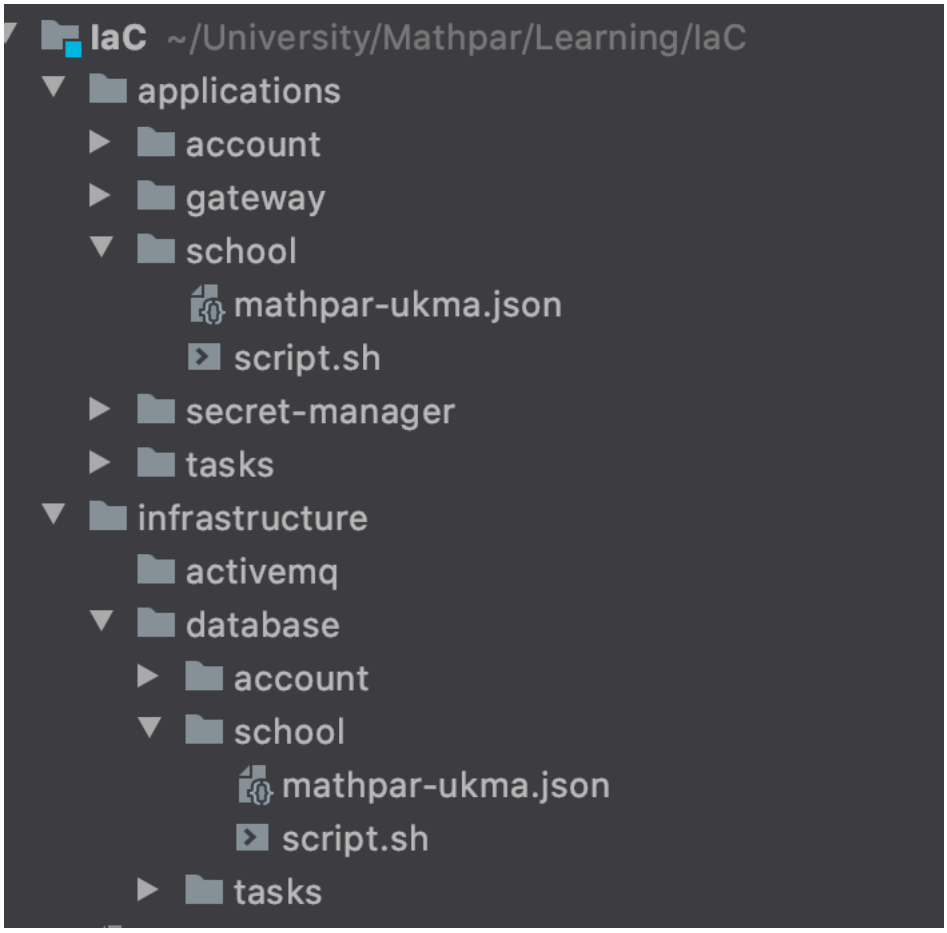


Рис 1.4.2. Структура репозиторія IaC.

Приклад фалу для підготовки команди розгортання:

```
networkName=$(jq -r ".networkName" < "$propertiesFile")
networkAlias=$(jq -r ".networkAlias" < "$propertiesFile")
containerName=$(jq -r ".containerName" < "$propertiesFile")
schemaName=$(jq -r ".schemaName" < "$propertiesFile")
VOLUME_BINDING=$(jq -r ".volumeBinding" < "$propertiesFile")

ROOT_PASSWORD="$passwordString"

echo "docker stop $containerName || true"
echo "docker rm $containerName || true"
echo "docker network inspect $networkName >/dev/null 2>&1 || docker network
create --driver bridge $networkName"
echo "docker run -d --network=$networkName --name $containerName --network-
alias $networkAlias -e 'MYSQL_ROOT_PASSWORD=$ROOT_PASSWORD' -e
'MYSQL_DATABASE=$schemaName' -v '$VOLUME_BINDING:/var/lib/mysql'
mysql:latest"
```

Mathpar Learning також реалізовує підходи B/G розгортання описані в розділі 1.3. Для реалізації даних підходів були визначені додаткові інструкції розгортання та розробки які дозволяють реалізацію підходу. Окрім того необхідні модифікації були зроблені до архітектурної складової. Розглянемо усі зміни проекту внесені для підтримки B/G розгортання:

- Правила по версіюванню відкритого функціоналу. Кожен з сервісів повинен версіювати відкритий функціонал. Заборонено вносити будь-які зміни до уже існуючих частин відкритого функціоналу, для модифікації необхідно створити копію функціоналу і модифікувати безпосередньо її. Коли всі споживачі перейдуть на нову версію функціоналу, стара версія може бути помічена як застаріла та видалена.
- Зміни до алгоритму розгортання бекенд сервісів. Під час розгортання спочатку створюється екземпляр нової версії сервісу. Проходить усе необхідне внутрішнє тестування щоб переконатись що розгортання пройшло успішно (фактично відбувається хелсчек, але можуть також проходити додаткові тестування, як наприклад використання смоук тестів). Якщо розгортання нового екземпляру проходить успішно, лоад балансер для відповідного сервісу починає відправляти трафік на новий екземпляр. Якщо все відбувається успішно і помилок не помічено,

старий екземпляр сервісу зупиняється, а згодом автоматично видаляється.

Таким чином під час розгоратння сервісів, користувачі не відчувають жодних змін оскільки кожен запит проходить успішно без необхідності оновлення сторінок/конфігурацій.

Велика увага під час розробки системи Mathpar Learning також приділялась тестуванню. Окрім юніт тестів для кожного з сервісів був розроблений спеціальний власний тестовий фреймворк з використанням відкритого фреймворку Cucumber-java. З його використанням були створена потужна тестова база яка дозволяє тестувати вже інтегрований, розгорнутий застосунок на певному конкретному середовищі для верифікації роботи конкретних частин функціоналу. Залежно від детальності тестування можна проводити:

- Поверхневе API тестування, яке перевіряє лише працездатність головних функціональних частин системи (створення аккаунтів, шкіл, робота з завданнями та планами тощо). Такий тип тестування проводиться найшвидше, але лише верифікує мінімальний набір функцій щоб переконатись що загалом застосунком можна користуватись без видимих помилок. Для тестування використовуються відкриті API ендпоінти, які використовуються клієнтським застосунком.
- Детальне API тестування, яке перевіряє працездатність усіх функціональних частин системи. Цей тип тестування включає в себе попередній тип тестування з більшим покриттям. Фактично в результаті даного тестування можна зробити висновок про те чи можна використовувати усі функції системи. Для тестування використовуються відкриті API ендпоінти, які використовуються клієнтським застосунком.

- Регресійне тестування, яке перевіряє не лише працездатність усіх функціональних частин системи, а й конкретні часткові випадки та потенційні/помічені раніше помилки. В результаті даного тестування можна зробити чіткий висновок про стан системи. В результаті того що дане тестування має найвище покриття воно є дуже точним і може використовуватись для остаточної валідації готовності застосунку для переходу в клієнтське (Production) середовище. Але саме через таке високе покриття кодової бази цей тип тестування відбувається найдовше
- UI тестування, яке відкриває віртуальний браузер і використовує його для верифікації коректності роботи клієнтського застосунку. Таке тестування є найближчим до клієнта, оскільки перевіряє напряму ту частину функціоналу з якою взаємодіє клієнт.

Важливим зауваженням також є те, що кожен тест в будь-якому тестовому наборі є абсолютно незалежним від інших тестів, порядку тестування і середовища. Будь-які ресурси необхідні для проведення тесту створюються самим тестом і це також відповідальність самого тесту зачищати будь-які тесові дані створені під час тестування. Таким чином будь-який набір тестів може бути використаний для тестування будь-якого середовища в будь-який момент часу. Жодних ручних дій по завершенню роботи тестів проводити не потрібно.

Приклад тестового сценарію виконаного з використанням бібліотеки Cucumber:

```
Scenario: Class can be populated with students and students could be removed
  When I try to create class with name "Test class 1"
  Then Request finished with status 200
  And I save result with key "class1"
  When I try to modify class "class1" and add students
    |Student|
    |Student2|
  Then Request finished with status 200
  When I try to get class "class1"
  Then Request finished with status 200
  And I verify that class has only following students
    |Student|
    |Student2|
  When I try to modify class "class1" and remove students
    |Student|
  Then Request finished with status 200
  When I try to get class "class1"
  Then Request finished with status 200
  And I verify that class has only following students
    |Student2|
```

Однак варто також зауважити, що у випадку поломок в механізмах створення/видалення аккаунтів, шкіл, профілів тощо існує ризик залишкових даних після проведення тестування, оскільки для обох типів операцій використовується не безпосередньо зв’язок з базою даних, а доступні публічні ендпоінти. Це зроблено з метою можливості запуску тестів з будь-якого пристрою без необхідності створювати захищений зв’язок з сервером/віртуальною мережею в яких розташовані застосунки та їх інфраструктура.

Весь вихідний код тестового фреймворку доступний за покликанням:
https://github.com/MathparLearningTeam/ApiTestFramework

Глава 2. Використання Mathpar Calculator

2.1. Створення навчального матеріалу за допомогою LaTeX інструментарію.

2.1.1. Створення стандартизованих національних підручників.

Коли ми говоримо про дистанційну освіту та електронне навчання, однією з найважливіших проблем постає спосіб подання матеріалу та завдань учням. При стандартній схемі освіти, матеріали подаються з використанням паперових матеріалів (книг, посібників, збірників тощо). В таких матеріалах автори можуть викладати свої ідеї максимально доступним способом завдяки тому що надрукований статичний матеріал є незмінним та завжди матиме вигляд наданий автором. Коли ми розглядаємо електронні матеріали, не можна забувати про відсутність цього фактора. Технології постійно змінюються і кожна зміна в технології може змінити вигляд подання матеріалу. Розглянемо для прикладу математичні формули. Нехай вони подаються з використанням HTML+CSS для відображення спецсимволів та позиціонування елементів. Розвиток цих технологій окрім створення нових правил також вносить зміни до існуючих для виправлення помилок, вразливостей та з інших можливих причин. В свою чергу веб-браузери завжди повинні підтримувати найновіші версії технологій і з плином часу застарівші правила залишені для зворотньої підтримки будуть видалені. Яскравим прикладом схожих видалень є досить гучний інцидент в якому веб браузер Google Chrome видалив підтримку використання Java Applets в одному з оновлень через їх вразливості. В зв’язку з цим при використанні HTML+CSS для подання формул ми можемо зіткнутись з ситуацією при якій відображення формул непередбачувано змінюється і, фактично, всі наші матеріали спотворюються.

Підсумовуючи вищеописане, нам потрібен чіткий, стандартизований спосіб для представлення електронних матеріалів. Розглянемо 2 потенційні способи розв’язання цієї проблеми:

- Використання зображень для подання матеріалів. У такому випадку електронний вигляд матеріалів є фактично скануванням паперових матеріалів. Але за такого підходу дуже поскладнюється створення унікальних матеріалів вчителем і практично унеможливлюється редагування існуючих. Навіть прості модифікації як зміна цифр або виправлення синтаксичних/логічних помилок в прикладах.
- Використання системи розмітки яка має чіткий визначений інтерпретатор для візуального відображення. При використанні даного способу, всі підручники зберігаються в текстовому форматі та специфічній розмітці, що дозволяє легко редагувати та створювати нові матеріали. Мінусами даного підходу є необхідність автора знати систему розмітки (високий поріг входу) та складність першої імплементації.

В межах проекту Mathpar Learning буде використовуватись другий підхід, оскільки він надає набагато більш обширні можливості. В якості системи розмітки буде використовуватись LaTeX. Це спричинено поширеністю даної системи розмітки, що понижує поріг входу для авторів підручників. Також завдяки поширеності системи розмітки існує досить багато технічних засобів з відкритим вихідним кодом для спрощення розробки.

Приклад тексту написаного з використанням розмітки LaTeX:

$ a^n • b^n = $\underbrace{a • a • ... • a}$ • $\underbrace{b • b • ... • b}$ = a • a • ... • a • b • b • ... • b = $\underbrace{(ab) • (ab) • ... •(ab)}$ = $(ab)^n$.

Даний текст транслюється в наступне візуальне відображення:

$$a^n \cdot b^n = \underbrace{a \bullet a \bullet \ldots \bullet a}_{} \cdot \underbrace{b \bullet b \bullet \ldots \bullet b}_{} = a \cdot a \cdot \ldots \cdot a \cdot b \cdot b \cdot \ldots \cdot b = \underbrace{(ab) \bullet (ab) \bullet \ldots \bullet (ab)}_{} = (ab)^n.$$

Рис 2.1.1.1. Візуальне відображення тексту написаного з використанням LaTeX розмітки

2.1.2. Створення електронного підручника вчителем або автором.

Оскільки головними наповнювачами системи матеріалами після створення початкової бази підручників будуть вчителі або автори підручників, нам потрібно створити чіткий та зручний спосіб для створення цих самих матеріалів або перенесення їх з паперового вигляду. Враховуючи, що електронні матеріали зберігаються в форматі LaTeX тексту, ми можемо користуватись уже існуючими засобами для створення та редагування таких текстів.

Але окрім того потрібно надати можливість вчителю або автору писати тексти в інтегрованому середовищі самого застосунку для уникнення залежностей від зовнішніх умов. Для цього необхідно надати можливість не лише писати текст матеріалу з використанням розмітки, але ще й валідувати та робити попереднє обрахування і відображення тексту в написаній розмітці.

Можливість писати текст та попереднє його відображення це завдання які легко реалізувати використовуючи можливості одного лише клієнтського застосунку, для цього вже існує досить багато різноманітних бібліотек та допоміжних матеріалів. Єдине питання яке залишається до клієнтського застосунку – зручність написання тексту матеріалу використовуючи розмітку. Оскільки матеріали зазвичай міститимуть велику кількість формул, необхідно надати можливість швидко переключатись між режимами набору тексту та спеціальних символів, а також додати систему доповнення формул. Це повинно сильно спростити та заохотити викладачів до створення електронних матеріалів.

Іншою частиною написання матеріалів є обрахунок значень. Фактично, система Mathpar Calculator дозволяє не просто переводити розмітку LaTeX у візуально скомпонований ряд, але й обраховувати значення змінних та формул зазначених в тексті. Саме завдяки цьому досягається інтерактивність лекційного матеріалу, яка була згадана раніше. Також система дозволяє

створювати зображення з використанням спеціальної розмітки яка задає графік формульно. Для отримання таких результатів, текст розмітки матеріалу повинен бути відправлений на сервер для опрацювання. Оскільки ця процедура є серверною, тобто набагато більш ресурсозатратною за попередню, використовувати її при кожній зміні тексту є досить нераціонально. Тому в якості рішення буде пропонуватись можливість переглянути кінцевий вигляд матеріалу (відрендерений на серверній частині) при натисканні на спеціальну кнопку.

2.2. Створення самостійних і контрольних робіт, що містять повні рішення, як прототипи рішень школяра у хмарному зошиті.

Окрім матеріалів для навчання, система також повинна надавати можливість представляти умови завдань для вирішення. Такими завданнями можуть бути як тестові завдання, так і завдання з розгорнутою відповіддю. Оскільки представлення задач в тестовому форматі це тривіальна задача реалізована вже неодноразово в багатьох рішеннях з відкритим вихідним кодом, розглядати її реалізацію в межах проекту Mathpar Learning ми не будемо.

Розглянемо створення задач з відкритою відповіддю. Задачі такого типу містять 2 компоненти:

- Умова
- Остаточна відповідь у формульному або цифровому вигляді

Варто розуміти, що задачі з відкритими відповідями не обов’язково містять цифрову відповідь, оскільки деякі з них можуть бути параметризованими, що вимагатиме дослідження можливих значень параметрів або інші параметризовані відповіді. В зв’язку з цим використовувати стандартні математичні операції для перевірки відповіді недостатньо.

Розглянемо можливість використання сервісу Mathpar Calculator для вирішення даної проблеми. Оскільки Mathpar Calculator дозволяє проводити

обрахунки над формульними виразами, ми можемо визначити відповідь на задачу з відкритою відповіддю як формульний вираз заданий мовою розмітки LaTeX. Таким чином будь-який вираз може бути зарахований у якості відповіді. Також таким чином можна відображати відповіді у зручному для перегляду форматі без необхідності додаткових взаємодій як і умови та лекційні матеріали. Таким чином, при створенні задачі в якості відповіді нам достатньо приймати вираз написаний мовою LaTeX та зберігати його в базі даних для подальшого порівняння з відповідями учнів. Окрім того, відповідь може бути задана не просто формульним рядком, а повноцінним покроковим розв'язком. Таким чином використовуючи сервіс Mathpar Calculator покрокове рішення може бути конвертоване до фінального формульного рядка для порівняння з відповідями студентів.

Умови задач фактично є аналогічними до прикладів задач у лекційних матеріалах. Оскільки приклади задач у матеріалах зберігаються як і решта лекційного тексту у вигляді LaTeX розміченого тексту, умови задачі будуть зберігатись аналогічно. Для відображення умови задачі ця сама розмітка буде використана для коректного відображення візуального відображення задачі.

2.3. Використання Mathpar Calculator, як робочого зошита для вирішення математичних завдань.

Оскільки ми розглядаємо можливість використання задач з відкритою відповіддю для оцінювання студентів, необхідно також надати їм всі необхідні умови для розв'язання цих самих задач. Для спрощення процесу використання системи варто намагатись максимально наблизити її до давно відомого і звичного способу розв'язання задач з відкритою відповіддю – використання чернеток для ведення записів. Коли задачі з відкритою відповіддю розв'язуються в паперовому вигляді, ми використовуємо чернетки для попередніх обрахунків, нотування ідей та часткових рішень, проведення складніших обрахунків тощо. Таким чином, якщо ми хочемо надати

можливість зручно розв’язувати задачі в електронному вигляді необхідно надати можливість емулювати функціонал чернеток. Враховуючи потужність комп’ютерів та швидкість їх обчислень, такий спосіб розв’язання задач може в результаті виявитись навіть зручнішим за класичний паперовий варіант.

Для емулювання електронної чернетки необхідно надати можливість стуентам як нотувати простий текст для збереження ідей та підходів до розв’язку, так і записувати формули або проводити проміжні обрахунки. LaTeX дозволяє нам як записувати текстові нотатки, так і проектувати формульні вирази, а сервіс Mathpar Calculator дозволяє переводити LaTeX вирази у візуальну форму та обраховувати їх. Таким чином, використання LaTeX для допомоги в обрахунках та розв’язанні задач дозволяє нам відтворити процес розв’язання задач в реальному часі.

Однак залишається проблема легкості використання такого функціоналу, - не кожен студент знає LaTeX та може з легкістю ним користуватись. Враховуючи цей факт, необхідно додатково розробити спосіб використовувати розмітку без необхідності її попередньо вивчати та звикати до її використання. Для цього необхідно розробити інтегроване середовище, яке дозволятиме використати зручний та зрозумілий інтерфейс для створення необхідної розмітки та задання формул.

2.4. Використання Mathpar Calculator, як засоба автоматичної перевірки правильності рішення, повторного рішення і демонстрації правильного рішення за запитом учня.

Важливою задачею яку повинен вирішувати сервіс Mathpar Learning є задача перевірки коректності відповіді на практичні завдання з відкритою відповіддю. На відміну від тестових задач, у яких перевірка правильності відповіді зводиться до порівняння номеру обраного варіанту, задачі з відкритою відповіддю не мають єдиного варіанту який можна просто порівняти на однаковість. А враховуючи що система дозволяє розміщати

задачі з відкритою відповіддю в формульній формі, необхідно розробити спосіб автоматичної перевірки коректності рішення навіть в таких випадках.

Для імплементації рішення цієї проблеми Mathpar Learning буде використовувати функціонал наданий сервісом Mathpar Calculator. Він надає можливість використовувати наступні функції:

- Виконувати операції написані в LaTeX розмітці
- Порівнювати два надані списки операцій на рівність

Друга функція є необхідною для розв'язання вищезгаданої проблеми. Використовуючи дану функцію ми можемо порівнювати відповіді студентів, які будуть надходити у вигляді LaTeX розміченого тексту з текстом такого ж формату, збереженим у базі даних в якості відповіді на задачу. Таким чином, ми зможемо дізнатись коректність наданої користувачем відповіді.

Під час виконання операції порівняння відбувається віднімання двох параметрів (фактично - тексту наданого студентом та тексту збереженого в базі даних). Віднімання двох виразів позбавляє нас проблем різності форматів правильних відповідей, оскільки рівнозначні вирази в результаті віднімання будуть давати 0. Таким чином, якщо при відніманні обох виразів ми отримаємо порожнє значення, це означатиме що відповіді рівнозначні. Таким чином ми зможемо перевіряти відповідь користувача на коректність.

Використовуючи такий підхід ми також можемо без жодних проблем зберігати історію наданих учнем/студентом відповідей, зберігаючи їх в якості LaTeX розміченого тексту в базі даних. Завдяки цьому Mathpar має приймати більше однієї спроби вирішення одного й того ж завдання. Це є дуже корисною функцією, коли в розрахунок беруться практичні завдання, які створені для навчання, а не перевірки знань і повинні все одно бути перевірені на коректність для коректності процесу навчання.

Оскільки коректні відповіді на задачі зберігаються в базі даних в якості розміченого тексту, студент також може отримати коректну відповідь у

візуальному стилі якщо в цьому буде потреба. Таким чином, якщо студент, наприклад, не знає як розв'язати завдання, він може «здатись». Коли студент здається, він не отримує балів за завдання, але має можливість дізнатись коректну відповідь для того аби дізнатись як правильно розв'язувати завдання. Такий функціонал є доступним для використання з обмеженим колом задач, оскільки правильна відповідь на екзаменаційні задачі не повинна поширюватись.

Рис 2.4.1. Приклад запиту для перевірки відповіді користувача

Приклад реалізації в проекті:

```
/**
 * This method checks if the solution provided as first argument correct
against the solution provided as the second parameter.
 * Both solutions need to be in LaTeX format
 * @param userSolution the solution which need to be compared (in LaTeX). In
scope of school this is the student's solution
 * @param databaseSolution the solution to which we compare (in LaTeX). In
scope of school this is the solution provided by the task's creator.
 * @return true if the userSolution is equals to databaseSolution. False
otherwise.
 */
public boolean isSolvedCorrectly(String userSolution, String
databaseSolution){
    try {
        var result = restTemplate.postForObject(urlPrefix + "/check", new
CheckSolutionPayload(userSolution, databaseSolution),
CheckSolutionResponse.class);
        if (result == null || result.error != null) throw new
InternalComputationException(result==null?"Empty response
returned":result.error);
        return result.result.equals(Constants.MATHPAR_CORRECT_ANSWER_KEY);
    } catch (HttpClientErrorException ex){
        throw new InternalComputationException(String.format("Request failed
with status %s, message: %s", ex.getStatusCode().value(), ex.getMessage()));
    } catch (Exception ex){
        throw new RuntimeException(String.format("Unexpected exception
occurred during checking solution %s compared to %s", userSolution,
databaseSolution), ex);
    }
}
```

2.5. Автоматичне збереження дій учня і правильності вирішення завдань в щоденнику, який доступний шкільної адміністрації, освітньої обласної та національної адміністрації.

Система Mathpar Learning повинна надавати можливість вчителям не лише проводити дистанційне або додаткове навчання для студентів, а ще й допомагати в оцінюванні їх можливостей, виставленні оцінок та інше. Тобто окрім можливості підбирати та надавати набори лекційних, практичних та екзаменаційних матеріалів, учителі повинні мати можливість проглянути успішність будь-якого учня по наданим матеріалам, його зусилля та час проведений над тими чи іншими завданнями.

Оскільки запис власне часових міток може призвести до морального тиску на студентів та негативно вплинути на їх зусилля, кількість часу витраченого на завдання повинна визначатись іншим способом.

Окрім того, оскільки вчителі часто мають одночасно працювати з величезною кількістю студентів, необхідно надати їм можливість швидко та в компактному вигляді повністю оцінити успішність будь-якого студента без необхідності розглядати детальну інформацію про кожне пройдене ним завдання.

Для подолання обох вищезазначених проблем була створена система рядкового подання успішності. Її суть полягає в кодуванні прогресу конкретного учня в конкретному наборі задач в якості єдиного рядку символів за заданими правилами. Такими правилами є:

- Якщо учень зміг розв’язати конкретну проблему з N спроби, то задача кодується в якості цифри 1-9 залежно від N (при N > 10 залишається цифра 9 як максимально можлива кількість спроб для статистики)
- Якщо учень зробив N спроб і не зміг правильно розв’язати задачу, то задача кодується літерою A-I (при N>10 залишається літера “I” як максимально можлива кількість спроб для статистики)

- Якщо учень зробив N спроб і здався (вирішив піддивитись коректну відповідь), то задача кодується літерою a-i (при N>10 залишається літера “i” як масимально можлива кількість спроб для статистики)

Таким чином, будь-яка задача подається у вигляді єдиного символу і ми маємо можливість подавати статистику по величезній кількості задач у якості однієї єдиної стрічки.

Це дозволяє нам не просто показати успішність учня вчителю по набору задач, а ще й вибудувати статистику учня по всім предметам за певний проміжок часу в компактному вигляді для адміністрації школи, або обласної/національної освітньої адміністрації.

Приклад реалізації сутності щоденника:

```
@Data
@Entity
public class StudentDiary {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public long id;
    @Column(name = "practice_id")
    public long practiceId;
    @Column(name = "result_string")
    public String resultString;
    @Column(name = "last_update")
    @Temporal(TemporalType.TIME)
    @UpdateTimestamp
    public Date lastUpdate;
}
```

Глава 3. Створення бази задач та завдань для використання в освітньому процесі

3.1. База навчальних матеріалів

Корисність системи Mathpar Learning проявляється не лише в її функціях для дистанційного та додаткового навчання, але ще й в базі попередньо набраних навчальних матеріалів. Такі матеріали дозволять вчителям швидко створювати власні навчальні плани без необхідності витрачати багато часу для створення власних матеріалів, або перенабирання існуючих. Це спростить роботу з системою і дозволить школам легше впроваджувати систему Mathpar Learning в навчальний процес.

Для створення цілої бази навчальних матеріалів, необхідно визначитись з джерелами лекційних матеріалів та способом їх імпортування у систему. Розглянемо обидві проблеми та можливість їх вирішення поокремо.

Джерела лекційних матеріалів. Оскільки Mathpar Learning створений лише для STEM дисциплін, для яких існують визначені державні рекомендації, визначення початкового джерела лекційних матеріалі не складає труднощів. Потрібно визначити мінімальний список необхідних матеріалів для покриття навчальних планів цільових класів. Визначений таким чином список лекційних матеріалів і буде мінімально необхідною базою матеріалів для викладачів. Система Mathpar Calculator має базу таких матеріалів збережених у LaTeX розмітці, що дозволяє нам використати їх в системі Mathpar Learning без необхідності додаткових ручних кроків.

Спосіб імпортування матеріалів у систему. Для імпортування у систему необхідно перевести наявні у системі Mathpar Calculator матеріали до формату який використовує система Mathpar Learning. Після цього достатньо виконати збереження конвертованих матеріалів до бази даних системи Mathpar Learning з використанням публічно доступних ендпоінтів. Таким чином постає потреба

в створенні утиліти, яка буде проводити процедуру конвертації з одного формату в інший та відправлятиме результат для збереження в базу даних. Оскільки всі матеріали збережені в системі Mathpar Calculator мають однорідний формат і збережені в JSON об’єктах, їх імпортування полягає в:

1. Зчитуванні об’єкту з бази даних Mathpar Calculator
2. Використання зчитаного JSON об’єкту для визначення автора, текстової частини, джерела та інших необхідних для Mathpar Learning системи значень
3. Створення тіла запиту для збереження об’єкту в Mathpar Learning
4. Відправка запиту для збереження об’єкту в Mathpar Learning

Таким чином початкова база даних підручників буде доступна для будь-якого вчителя в системі, що спростить роботу з нею та дозволить створювати ефективні та корисні навчальні плани.

Приклад реалізації процесу імпорту:

```
ResultSet resultSet = databaseConnector.getResultSet(batchSize, 0);

try {
   while (!resultSet.isAfterLast()) {
      List<DB_Row> rows = DB_Row.parseRows(resultSet, batchSize);
      List<SkippedEntity> skippedEntities =
recordProcessor.processDbRows(rows);
      handleSkippedEntities(skippedEntities, exportPrefix);
   }
resultSet.close();
databaseConnector.wrapUp();
} catch (Exception e){
   throw new RuntimeException("Unhandled exception during processing the
result was thrown", e);
}
```

3.2. База самостійних та контрольних робіт

Оскільки одних лишень навчальних матеріалів для повноцінного та ефективного дистанційного навчання недостатньо, проект Mathpar Learning також надає можливість створювати практичні завдання для використання в самостійних та екзаменаційних роботах. Але практичні завдання містять більше елементів та потребують більших затрат часу для створення в зв’язку з

цим. Саме тому необхідність створити базу практичних завдань для спрощення створення курсів учителям або адміністрації є навіть більш важливим завданням ніж створення бази лекційних матеріалів.

Сервіс Mathpar Calculator окрім попередньо створеної бази лекційних матеріалів також має достатню кількість практичних матеріалів у форматі LaTeX текстів. Задачі створені в ньому також містять необхідні розв'язання та відповіді тому є підходящими до використання в сервісі Mathpar Learning.

Для імпортування таких задач в систему Mathpar Learning можна використовувати аналогічний підхід до імпортування лекційних матеріалів. Єдина різниця між лекційними матеріалами та практичними завданнями полягає в кількості полів необхідних для подання завдання в системі, тому використання утиліти розробленої для імпортування лекційних матеріалів є досить очевидним рішенням. Таким чином, створення бази даних

3.3. Навчальні плани

3.3.1 Створення бази стандартизованих національних планів проходження навчальних дисциплін.

Маючи базу лекційних матеріалів та практичних завдань, система Mathpar Learning може надавати можливість з легкістю створювати освітній процес. Але для його систематизації та контролю не достатньо просто надавати матеріали, необхідно також мати можливість задавати структуру, наповнення та систему контролю навчального процесу. Для вирішення цієї проблеми було введення поняття навчального плану. Навчальний план – це збірка лекційних матеріалів, практичних та екзаменаційних завдань які логічно об'єднані в схему навчання для досягнення певної мети. Наприклад курс для 7 класу містить в собі всі необхідні лекційні матеріали, практичні та екзаменаційні роботи які потрібно пройти для успішного отримання знань за 7-й клас.

Кожен план складається з блоків, в які можуть входити матеріали, практичні та екзаменаційні завдання. Фактично блок є універсальною структурною одиницею без обмежень. Розглянемо структуру блоку:

- Кожен блок обов'язково містить дату та час початку та може мати дату та час завершення. Таким чином кожен блок може відображати найрізноманітніші логічні згурпування завдань. Блок може відображати навчальний тиждень, обмежений курс, або й взагалі бути єдиним на курс якщо логічне розбиття не є потрібним.
- Кожен блок обов'язково містить список лекційних матеріалів. Такий список може містити від 0 до 1000 навчальних матеріалів для студентів. Таким чином один блок може містити як усі матеріали курсу, так і жодного матеріалу у випадку якщо блок є винятково екзаменаційним.
- Кожен блок обов'язково містить список практичних завдань з умовами аналогічними до списку лекційних матеріалів.
- Кожен блок може містити екзаменаційне завдання, або не містити його. В межах блоку екзаменаційне завдання – це фінальне випробування яке студент повинен пройти для успішного завершення блоку. Робити кілька фінальних випробувань для оцінки вмінь студентів в межах одного блоку немає потреби, тому у випадку якщо потрібно провести кілька оцінювань успішності студента правильним підходом буде створення кількох блоків.
- Кожен блок може містити назву та опис для спрощення усвідомлення його призначення та подачі теми учням.

У класичному та рекомендованому розумінні, один блок повинен представляти одну навчальну тему навчального плану. Таким чином в межах блоку може бути довільна кількість матеріалів на цю тему (обов'язкових та додаткових, для самостійного опрацювання), довільна кількість наборів практичних завдань різних рівнів складності а також єдине екзаменаційне

завдання для обов’язкових тем. Це також дозволяє створення блоків опційних тем, які не повинні впливати на загальну оцінку.

3.3.2. Створення учителем бази планів проходження навчальної дисципліни для навчальної групи.

Система Mathpar Learning має попередній набір станадартизованих навчальних планів рекомендованих освітньою адміністрацією. Їх використання зводиться до тривіального призначення уже існуючого шаблону навчального плану до навчальної групи та початку проходження матеріалів. Але інколи з’являється потреба вносити корективи до навчальних планів у зв’язку з іншим підходом до викладання, власними джерелами інформаціїї тощо. В зв’язку з цим система Mathpar Learning надає можливість гнучкої модифікації та створення власних планів вчителями загальноосвітніх закладів.

Оскільки стандартизовані освітні плани не повинні бути змінені вчителями конкретних шкіл, адже вони є рекомендаційними для всіх, прямі модифікації цих планів під потреби конкретної школи, навчальної групи або викладача не є можливими. Кожен стандартизований план може бути змінений лише його автором (освітньою адміністрацією). Зважаючи на це, в межах системи була розроблена можливість поверхневого копіювання навчального плана, під час якої втрачається можливість отримувати зміни навчального плану від автора, але надається можливість будь-яких модифікацій плану напряму. Такі плани зберігаються в базі даних як приватні плани конкретних шкіл/аккаунтів (викладачів) для можливості їх подальшого перевикористання в якості шаблону в межах тієї ж школи/курсів того ж аккаунту.

Таким чином будь-який викладач має можливість створювати власну базу навчальних планів залежно від власних потреб. Такі бази навчальних планів створені вчителями нададуть більше гнучкості при роботі з системою а також спростять роботу з нею.

Приклад сутності навчальних планів:

```
@Data
@NoArgsConstructor
@Entity(name = "plans")
public class SchoolPlan {
    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;
    @Column(name = "name")
    private String name;
    @Column(name = "description")
    private String description;
    @Column(name = "school_id")
    private long school;
    //Profile id of an author of this plan
    @Column(name = "author_id")
    private long authorId;

    @OneToMany(mappedBy = "plan_id")
    List<PlanBlock> blocks;

    public SchoolPlan(SchoolPlanTemplate template){
        this.name = template.getName();
        this.description = template.getDescription();
    }
}
```

```
@Data
@Entity
public class PlanBlock {
    @Id
    private long id;

    @JoinTable(name = "plan_id")
    private long plan;

    @Column(name = "time_start")
    @Temporal(TemporalType.TIME)
    private Date timeStart;
    @Column(name = "time_end")
    @Temporal(TemporalType.TIME)
    private Date timeEnd;

    @Column(name = "title")
    private String title;
    @Column(name = "description")
    private String description;

    @OneToMany(mappedBy = "block_id")
    private List<Exam> exams;
    @OneToMany(mappedBy = "block_id")
    private List<Lecture> lectures;
    @OneToMany(mappedBy = "block_id")
    private List<Practice> practices;
}
```

3.4. Створення бази "щоденників" для всіх учнів. Автоматичне заповнення, збереження і доступність бази "щоденників".

Виконання практичних та екзаменаційних завдань студентами є важливою частиною навчального процесу в системі Mathpar Learning. Вони надають можливість студентам відточувати свої навички та оцінювати власні уміння. Але окрім цього, виконання практичних та екзаменаційних завдань надає можливість вчителям та батькам оцінити успіхи учня в навчанні. Для цього створюється система «щоденників» - збірки віртуальних статистичних даних по учню. Кожен учень в системі Mathpar Learning має такий віртуальний «щоденник» який зберігає в собі будь-які дії та успіхи учня та надає можливість дізнатись прогрес виконання завдань вчителям або адміністрації.

Оскільки система Mathpar Learning є повністю автоматизованою, включаючи перевірку будь-якого типу завдань, система щоденників також повинна бути автоматизованою. Таким чином у вчителів відпадає необхідність монотонної рутинної роботи по заповненню звітностей кожного зі студентів і залишається більше часу та натхнення для творчого подання матеріалу і покращення навчального процесу.

Таким чином в щоденнику будь-якого учня будуть збережені будь-які його спроби проходити практичні завдання в якості закодованих символами рядків (детальніше про це можна прочитати в розділі 2.5) а також результати його екзаменаційних робіт. Завдяки міткості електронних носіїв інформації, такі компактні бази щоденників можуть зберігатись для будь-якого учня протягом усього його процесу навчання і в архівному режимі навіть опісля завершення навчання. Також електронна система щоденників дозволяє легку демонстрації успіхів учня будь-якому користувачу з доступом до відповідного щоденника. Таким користувачем може бути сам учень, викладач, адміністрація школи або будь-яка інша освітня адміністрація.

3.5. Перспективи організації національного моніторингу навчального процесу на основі бази "щоденників".

Завдяки легкодоступній базі щоденників також є можливість організації обширного та детального моніторингу навчального процесу будь-якого навчального закладу, або навіть будь-якої вибірки навчальних установ. Завдяки поданню основного прогресу учня (практичних завдань) у якості рядків існує можливість створення величезного скомпонованого рядка успішності будь-якої вибірки студентів і аналізу цього рядка електронними засобами.

Наприклад існує легка можливість для обласної освітньої адміністрації взяти вибірку всіх учні 11-х класів певної області, отримати сумарну вибірку усіх їх щоденників і підрахувати загальну кількість символів у скомпонованому рядку для отримання чіткого уявлення про складість матеріалу, старанність учнів та навантаження навчального плану завдяки підрахунку кількості символів на студента. Окрім аналізу практичних завдань також є можливість легко отримати вибірку фінальних оцінок студентів за певний блок завдань використовуючи записи про результати їх екзаменаційних робіт.

Такий аналіз не обмежується лише обласними адміністраціями, моніторинг може проводитись над найрізноманітнішими вибірками. Адміністрація освітнього закладу може проводити моніторинг успішності учнів, міська адміністрація може проводити аналіз успішності по місту та по конкретним школам тощо. Використання системи електронних щоденників відкриває нові підходи до роботи та аналізу навчального процесу в реальному часі. Завдяки хмарній природі сервісу Mathpar Learning такі вибірки та аналізи можна проводити в будь-який момент навчального процесу для швидкого реагування, коректування та покращення якості навчання.

Глава 4. Інтеграція хмарних обрахунків в освітній процес математики, фізики, хімії

4.1. Інтеграція хмарних обрахунків в освітній процес математики

Mathpar Learning в першу чергу орієнтується на інтегрування хмарних технологій та обчислень в освітній процес STEM дисциплін. Розглянемо спосіб інтегрування хмарних обрахунків у освітній процес математики. Як було згадано раніше, провайдером хмарних обрахунків для Mathpar Learning буде сервіс Mathpar Calculator. Він надає можливість обраховувати значення виразів написаних у розмітці LaTeX. Такі вирази можуть містити як базові арифметичні операції накшталт додавання, віднімання, ділення та множення, так і більш складні операції, як логарифми, тригонометричні операції, рівняння, інтеграли, похідні тощо.

Під час освітнього процесу вивчення математики, студенти повинні не лише отримувати відповіді до вказаних формульних виразів, але й вміти їх розв’язувати. Це вимагається не для запам’ятовування алгоритмів знаходження тих чи інших виразів або константних значень, а в першу чергу для навчання студентів логічно підходити до завдань, розвивати вміння пошуку різних підходів до розв’язання схожих задач, уміння правильно та детально аналізувати умови задач. Але одночасно з цим постійне повторення одних і тих самих складних операцій для отримання відповіді після того як студент уже вивчив та отримав достатньо практичних навичок у розв’язанні операції призводить до зниження інтересу до навчання та зниженню мотивації розв’язувати задачі.

Зважаючи все вищезгадане, використання хмарних обрахунків у освітньому процесі з математики повинно полягати в можливості спростити обрахунки тих функцій які вже були вивчені студентом, але не дозволяти використовувати обрахунки того що ще не було вивчено. Завдяки обширному

набору функцій сервісу Mathpar Calculator ми можемо забезпечити студента інструментарієм для автоматичного розв’язання практично будь-яких функцій. Відповідно, для того аби освітній процес був більш корисним та коректним, нам необхідно обмежити кількість функцій які може використовувати студент під час проходження того чи іншого матеріалу.

Завдяки такому підходу ми зможемо дуже спростити та зацікавити студентів у розв’язанні складних задач завдяки мінімізації механічної, рутинної роботи, але досі зберегти можливість вивчати та практикувати розв’язання тих чи інших функцій завдяки штучному обмеженню функціоналу хмарних обчислень.

4.2. Інтеграція хмарних обрахунків в освітній процес фізики

Під час вивчення фізики учень використовує математичні підходи для аналізу тих чи інших практичних задач. Оскільки акцент фізики зміщається з уміння розв’язувати ті чи інші математичні функції на знання фізичних формул, розуміння фізичних явищ та процесів і уміння інтерпретувати практичні задачі в фізичному контексті, хмарні обрахунки можуть зіграти дуже важливу та корисну роль для покращення освітнього процесу фізики.

Фізичні задачі можуть використовувати як складні математичні функції, такі як інтегрування та диференціювання, так і графічні аспекти математики, такі як декартові площини. Для обрахування складних математичних функцій дуже добре підходить використання хмарних обрахунків, які надають нам можливість підвищити точність обрахунків та сконцентруватись на вирішенні фізичної задачі. На відміну від необхідності обмежувати використання хмарних обрахунків для освітнього процесу математики, під час інтегрування їх у фізику обмежувати нічого потреби немає. Натомість потрібно надавати повний функціонал для всіх потенційних задач. Окрім того, сервіс Mathpar Calculator надає можливість працює з 2Д графікою (графічним відображенням

функцій для аналізу). Використання цього функціоналу дозволяє ще більше полегшити та покращити процес вивчення матеріалів фізики.

4.3. Інтеграція хмарних обрахунків в освітній процес хімії

Під час вивчення хімії, аналогічно до освітнього процесу фізики, математичні обрахунки є необхідними для розв'язання задач та розуміння хімічних процесів. Але використання математичних виразів та формул не є головним акцентом під час вивчення хімії. Натомість учню необхідно мати високу точність обрахунків та проводити ланцюги операцій для отримання фінальних значень та відповідей.

Враховуючи це, інтегрування хмарних обчислень в процес вивчення хімії є корисним доповненням. Точні обрахунки виконані з використанням хмарних обчислень є швидким та ефективним способом отримувати високу точність відповіді з мінімальними затратами часу. Окрім того, завдяки тому що сервіс Mathpar Calculator дозволяє визначення змінних в оброблюваному тексті, ми можемо без додаткових поскладнень спростити процес отримання відповіді на задачу.

Таким чином, інтегрувавши хмарні обчислення в освітній процес хімії ми маємо можливість підвищити ефективність студента, зменшити шанс на механічну помилку та підвищити точність обрахунків, при цьому не втрачаючи ефективності в навчанні та отриманні нового матеріалу.

Висновки

В результаті виконання дипломної роботи був проведений аналіз, розглянуті можливості інтеграції хмарних технологій та хмарних обчислень в сучасну STEM освіту для покращення її якості та ефективності. Окрім того був розроблений продукт який дозволяє легко інтегрувати вищезгадані технології та підходи в будь-який навчальний проект. Розглянемо отримані результати:

- Розроблені рекомендації по впровадженню мікросервісної архітектури в навчальні проекти та поданий приклад реалізації
- Розглянута можливість впровадження підходів B/G розгортання та подані приклади реалізації
- Розроблені та впроваджені процедури для CI/CD процесів
- Створений проект з відкритим вихідним кодом який реалізовує всі вищеописані підходи
- Розглянута можливість інтеграції хмарних обчислень в освітні процеси математики, фізики та хімії
- Розроблені та описані способи створення навчальних матеріалів та навчальних планів
- Створені власні засоби автоматизованого тестування функціоналу, юніт тести та інтегровані тести

Використовуючи отримані результати та розроблені програмні застосунки, наступним кроком буде їх розгортання в постійному середовищі для використання та проведення практичних тестів для отримання зворотного зв’язку і покращення проекту.

Література

1. Carnell J. Spring Microservices in Action / John Carnell.. – 384 с.
2. Microservice Architecture: Aligning Principles, Practices, and Culture / N.Nadareishvili, R. Nadareishvili, M. Matt, A. Mike.. – 146 с.
3. Microservices architecture style [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices.
4. Микросервисы (Microservices) [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://habr.com/ru/post/249183/.
5. Daniels R. Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale / R. Daniels, J. Davis., 2016. – 410 с.
6. Smart J. Jenkins: The Definitive Guide / John Smart., 2011. – 380 с.
7. Using Blue-Green Deployment to Reduce Downtime and Risk [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://docs.cloudfoundry.org/devguide/deploy-apps/blue-green.html#:~:text=Blue%2Dgreen%20deployment%20is%20a,live%20and%20Green%20is%20idle..
8. Matthias K. Docker: Up & Running: Shipping Reliable Containers in Production / K. Matthias, S. P. Kane., 2018. – 352 с.
9. Laster B. Jenkins 2: Up and Running: Evolve Your Deployment Pipeline for Next Generation Automation / Brent Laster., 2018. – 606 с.
10. Nedelcu C. Nginx HTTP Server: Adopt Nginx for Your Web Applications to Make the Most of Your Infrastructure and Serve Pages Faster Than Ever / Clement Nedelcu., 2010. – 327 с.
11. Tooke S. The Cucumber Book: Behaviour-Driven Development for Testers and Developers 2nd Edition / S. Tooke, H. Aslak, W. Matt., 2017. – 336 с.
12. Maven Release Plugin [Електронний ресурс] – Режим доступу до ресурсу: https://maven.apache.org/maven-release/maven-release-plugin/.

13. Jenkins User Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://www.jenkins.io/doc/.
14. Building REST services with Spring [Електронний ресурс] – Режим доступу до ресурсу: https://spring.io/guides/tutorials/rest/.
15. Github: Publishing a package [Електронний ресурс] – Режим доступу до ресурсу: https://help.github.com/en/packages/publishing-and-managing-packages/publishing-a-package.
16. Малашонок Г.І. Хмарна математика MathPartner в Києво-Могилянській академії.
17. Наукові записки НаУКМА. Комп'ютерні науки. 2017. Т. 198. С. 27–35
18. Malaschonok G. I. Mathpar Language Guide. Tambov : Publishing House of TSU, 2013. 125 p.
19. Malaschonok G. I. MathPartner Computer Algebra. Programming and Computer Software. 2017. Vol. 43, No. 2. P. 112–118.