Міністерство освіти і науки України Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №3

Технології розробки програмного забезпечення

«Основи проектування розгортання»

«Веб-браузер»

Виконав:
студент групи ІА-33
Мартинюк Ю.Р.

Перевірив:
Мягкий Михайло Юрійович

Київ 2025

Тема: Основи проектування розгортання

Мета: Навчитися проєктувати діаграми розгортання та компонентів для системи що проєктується, а також розробляти діаграми взаємодії, а саме діаграми послідовностей, на основі сценаріїв зроблених в попередній лабораторній роботі.

Зміст

Завдання

• Ознайомитись з короткими теоретичними відомостями.

• Проаналізувати діаграми створені в попередній лабораторній роботі а також тему системи та спроєктувати діаграму розгортання використання відповідно до обраної теми лабораторного циклу.

• Розробити діаграму компонентів для проєктованої системи.

• Розробити діаграму розгортання для проєктованої системи.

• Розробити як мінімум дві діаграми послідовностей для сценаріїв прописаних в попередній лабораторній роботі.

• На основі спроєктованих діаграм розгортання та компонентів доопрацювати програмну частину системи. Реалізація системи, додатково до попередньої реалізації, повинна містити як мінімум дві візуальні форми. В системі вже повинен бути повністю реалізована архітектура (повний цикл роботи з даними від вводу на формі до збереження їх в БД і подальшій виборці з БД та

відображенням на UI).

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму розгортання з описом, діаграму компонентів системи з описом, діаграми послідовностей, а також вихідний код системи, який було додано в цій лабораторній роботі.

Теоретичні відомості

Діаграми компонентів — показують модулі (компоненти), їхні залежності і артефакти (.jar, таблиці, html). Використовуються для логічного/фізичного поділу системи.

Діаграма компонентів UML є представленням проєктованої системи, розбитої на окремі модулі. Залежно від способу поділу на модулі розрізняють три види діаграм компонентів:

• логічні;

• фізичні;

• виконувані.

Коли використовують логічне розбиття на компоненти, то у такому разі проєктовану систему віртуально уявляють як набір самостійних, автономних модулів (компонентів), що взаємодіють між собою.

Діаграми розгортання — показують фізичні вузли (devices) і середовища виконання (execution environments), на яких розгортаються артефакти; зв’язки зазначають протоколи (HTTP, SQL/ODBC).

Діаграми розгортання представляють фізичне розташування системи, показуючи, на якому фізичному обладнанні запускається та чи інша складова програмного забезпечення.

Головними елементами діаграми є вузли, пов'язані інформаційними шляхами. Вузол (node) – це те, що може містити програмне забезпечення. Вузли бувають двох типів. Пристрій (device) – це фізичне обладнання: комп'ютер або пристрій, пов'язаний із системою. Середовище виконання (execution environment) – це програмне забезпечення, яке саме може включати інше програмне забезпечення, наприклад операційну систему або процес-контейнер (наприклад,

вебсервер).

Діаграми послідовностей — моделюють часову послідовність повідомлень між акторами/об’єктами (HTTP POST/GET: Browser – Server DB – Browser).

Діаграма послідовностей (Sequence Diagram) – це один із типів діаграм у моделюванні UML (Unified Modeling Language), який використовується для моделювання взаємодії між об'єктами системи у певній послідовності часу. Вона відображає, як об'єкти обмінюються повідомленнями, показуючи порядок і логіку виконання операцій. Діаграма складається з таких основних елементів:

Актори (Actors): Зазвичай позначаються піктограмами або назвами. Це користувачі чи інші системи, які взаємодіють із системою. Актори можуть бути

Об'єкти або класи: Розміщуються горизонтально на діаграмі. Вони позначаються прямокутниками з іменем об'єкта або класу під прямокутником. Кожен об'єкт має «життєвий цикл», який представлений вертикальною пунктирною лінією (лінія життя).

Повідомлення: Це лінії зі стрілками, які з'єднують об'єкти. Вони показують передачу повідомлень чи виклик методів. Стрілка може бути синхронною (звичайна стрілка) або асинхронною (лінія з відкритим трикутником) та з пунктирною лінією, що показує повернення результату.

Активності: Вказують періоди, протягом яких об'єкт виконує певну дію. На діаграмі це позначається прямокутником, накладеним на лінію життя. Контрольні структури: Використовуються для відображення умов, циклів або альтернативних сценаріїв. Наприклад, блоки "alt" (альтернатива) або "loop" (цикл).

Тема проєкту

6. Web-browser (proxy, chain of responsibility, factory method, template method, visitor, p2p) Веб-браузер повинен мати можливість зробити наступне: мати адресний рядок для введення адреси сайту, переміщатися і відображати структуру html документа, переглядати підключений javascript та css файли, перегляд всіх підключених ресурсів (зображень), коректна обробка відповідей з сервера (коди відповідей HTTP) – переходи при перенаправленнях, відображення сторінок 404 і 502/503.

Хід роботи

Діаграма розгортання

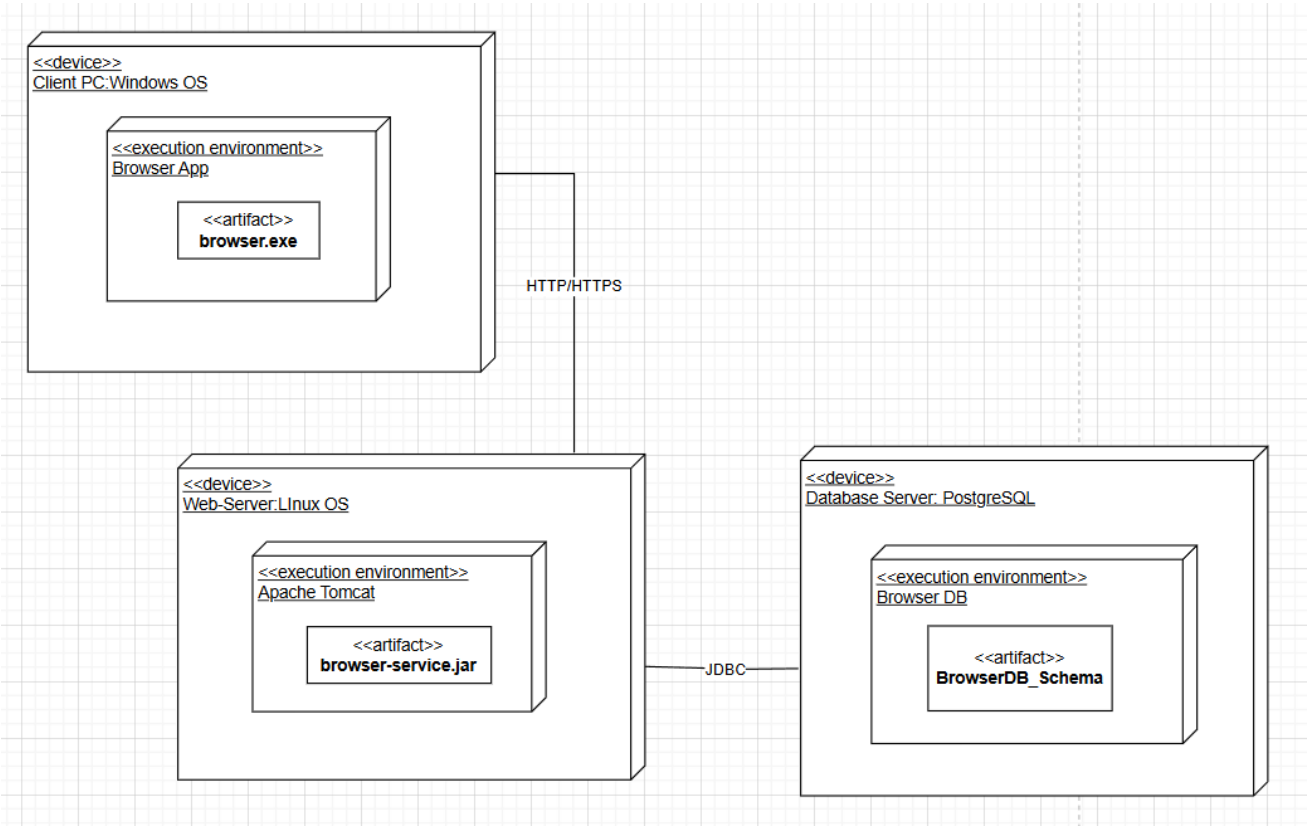

Рис.1 – Діаграма розгортання

Опис діаграми розгортання

1. Вузол «Client PC» (Клієнтська сторона)
Цей вузол представляє кінцевий пристрій користувача, на якому безпосередньо виконується клієнтський додаток.

- Пристрій (<<device>>): Client PC під управлінням операційної системи Windows OS.
- Середовище виконання (<<execution environment>>): Browser App, що є процесом операційної системи, який забезпечує запуск та функціонування клієнтського додатку.
- Артефакт (<<artifact>>): browser.exe. Це скомпільований нативний виконуваний файл, що містить всю клієнтську логіку: графічний інтерфейс користувача, механізм рендерингу веб-сторінок, обробку запитів користувача та взаємодію з віддаленим сервером.

2. Вузол «Web-Server» (Серверна сторона)
Цей вузол відповідає за обробку бізнес-логіки, керування даними користувачів та слугує проміжною ланкою між клієнтом та базою даних.

- Пристрій (<<device>>): Web-Server під управлінням серверної операційної

системи Linux OS.
- Середовище виконання (<<execution environment>>): Apache Tomcat. Це контейнер сервлетів та веб-сервер, який керує життєвим циклом Java-додатку та обробляє HTTP-запити.
- Артефакт (<<artifact>>): browser-service.jar. Це розгортуваний Java-архів, який містить скомпільовані серверні класи. Він реалізує API для взаємодії з клієнтом та інкапсулює логіку роботи з даними (репозиторії для користувачів, закладок, історії).

3. Вузол «Database Server» (Рівень даних)
Цей вузол призначений для персистентного (довготривалого) зберігання всієї системної інформації.
- Пристрій (<<device>>): Database Server. Це спеціалізований сервер, оптимізований для роботи систем управління базами даних (СУБД).
- Середовище виконання (<<execution environment>>): Browser DB. Це система управління реляційними базами даних (PostgreSQL), що відповідає за цілісність, зберігання та доступ до даних.
- Артефакт (<<artifact>>): BrowserDB_Schema. Даний артефакт представляє собою розгорнуту схему бази даних, що включає таблиці (Users, Bookmarks, HistoryItem тощо), їхні зв'язки, індекси та обмеження цілісності.

Комунікаційні Протоколи
- Зв'язок Client PC - Web-Server (HTTP/HTTPS): Взаємодія між клієнтською та серверною частинами здійснюється за стандартним веб-протоколом HTTP/HTTPS. Клієнт надсилає запити на сервер для автентифікації, синхронізації закладок, історії та інших даних, а сервер повертає відповіді у стандартизованому форматі (наприклад, JSON).
- Зв'язок Web-Server - Database Server (JDBC): Сервер додатку взаємодіє з базою даних за допомогою технології JDBC (Java Database Connectivity). Цей інтерфейс дозволяє серверній логіці виконувати SQL-запити до бази даних для виконання операцій створення, читання, оновлення та видалення даних (CRUD).

Діаграма компонентів

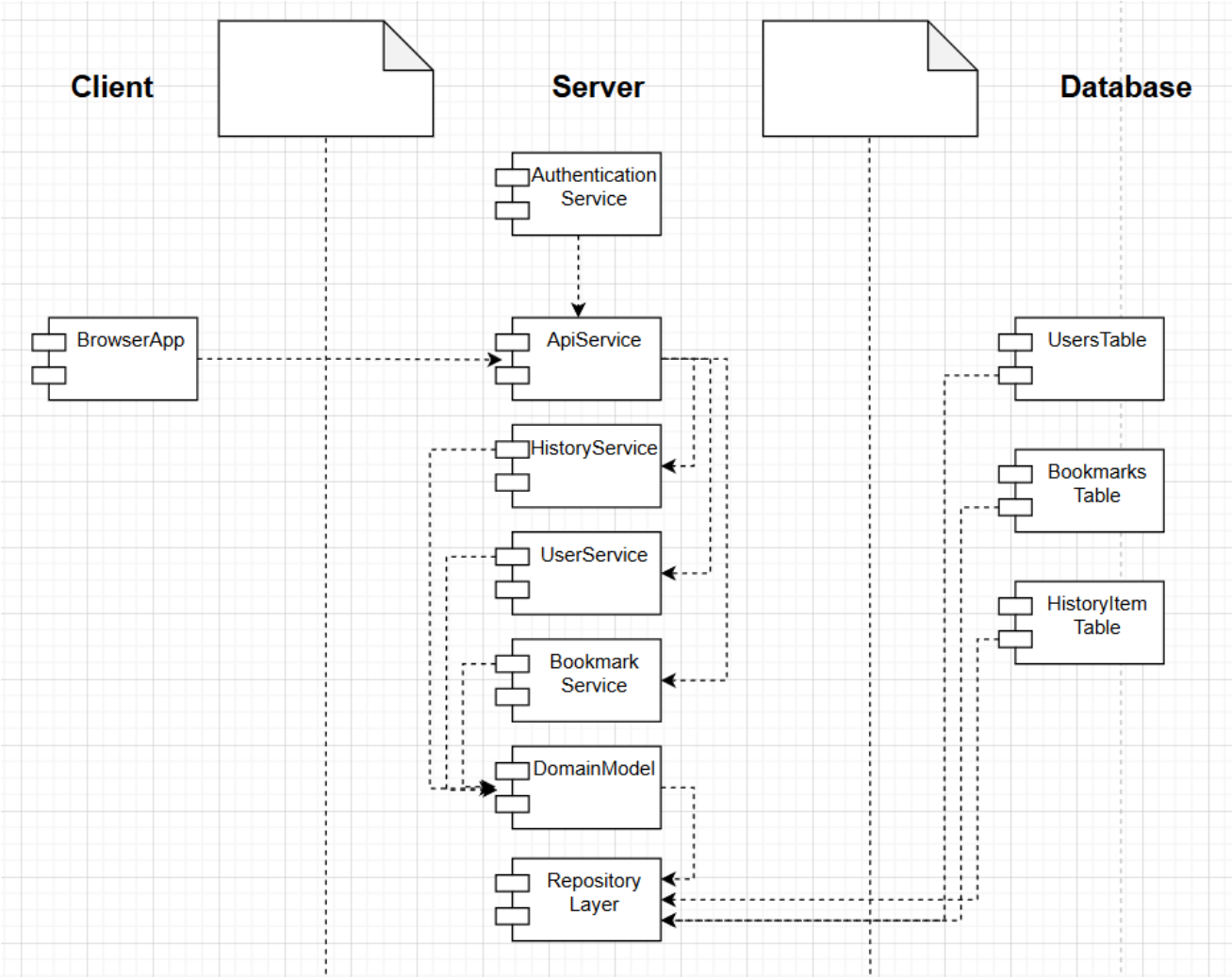

Рис.2 – Діаграма компонентів

Опис діаграми компонентів

Рівень Клієнта (Client)

- BrowserApp: Компонент, що являє собою клієнтський додаток. Він відповідає за користувацький інтерфейс, обробку взаємодії з користувачем та ініціалізацію запитів до серверної частини через ApiService.

Рівень Сервера (Server)

Серверний рівень реалізує всю основну бізнес-логіку та логіку доступу до даних.

- ApiService: Виконує роль API Gateway (шлюзу) або Фасаду. Це єдина точка входу для всіх запитів від BrowserApp. Ключовою відповідальністю компонента є оркестрація, що включає первинну обробку запиту, виклик сервісу автентифікації для перевірки прав доступу, та подальше делегування запиту відповідному бізнес-сервісу.

- AuthenticationService: Спеціалізований компонент, повністю ізольований від бізнес-логіки, який відповідає виключно за процеси автентифікації та авторизації. ApiService залежить від цього компонента для перевірки безпекових даних (наприклад, токенів доступу) перед виконанням будь-яких операцій.
- UserService, BookmarkService, HistoryService: Компоненти, що реалізують конкретну бізнес-логіку згідно з принципом єдиної відповідальності. Кожен сервіс оперує своєю доменною областю, використовуючи DomainModel для представлення даних та RepositoryLayer для їх збереження.
- DomainModel: Компонент, що містить визначення всіх бізнес-сутностей системи (класи User, Bookmark тощо). Він забезпечує уніфіковану структуру даних, яка використовується на сервісному рівні та рівні доступу до даних.
- RepositoryLayer: Шар абстракції доступу до даних, що реалізує шаблон "Repository". Він інкапсулює всю логіку взаємодії з базою даних (SQL-запити), надаючи бізнес-сервісам простий та зрозумілий інтерфейс для управління персистентністю даних.

Рівень Бази Даних (Database)

- UsersTable, BookmarksTable, HistoryItemTable: Компоненти, що символізують таблиці в реляційній базі даних. Вони є кінцевою точкою для зберігання даних, і з ними безпосередньо взаємодіє лише RepositoryLayer.

Загальний потік взаємодії

1. Клієнт (BrowserApp) надсилає запит до єдиної точки входу — ApiService.
2. ApiService отримує запит і першочергово звертається до AuthenticationService для перевірки прав доступу.
3. Після успішної автентифікації ApiService делегує запит відповідному бізнес-сервісу (наприклад, UserService).
4. UserService виконує необхідну бізнес-логіку, використовуючи сутності з DomainModel, та звертається до RepositoryLayer для маніпуляції даними.
5. RepositoryLayer трансформує запит у SQL та взаємодіє з відповідними таблицями (UsersTable) у базі даних.

Рівень Бази Даних (Database)

- UsersTable, BookmarksTable, HistoryitemTable: Ці компоненти

символізують фізичні структури зберігання даних (таблиці) у реляційній базі даних. RepositoryLayer напряму залежить від них для маніпуляції даними.

Загальний потік взаємодії

1. Запит від користувача надходить з BrowserApp до ApiService.
2. ApiService перенаправляє запит до відповідного сервісу (наприклад, BookmarkService).
3. Сервіс використовує сутності з DomainModel для роботи з даними та викликає RepositoryLayer для їх збереження чи отримання.
4. RepositoryLayer формує та виконує SQL-запити до відповідних таблиць (BookmarksTable) у базі даних.

Діаграми послідовностей

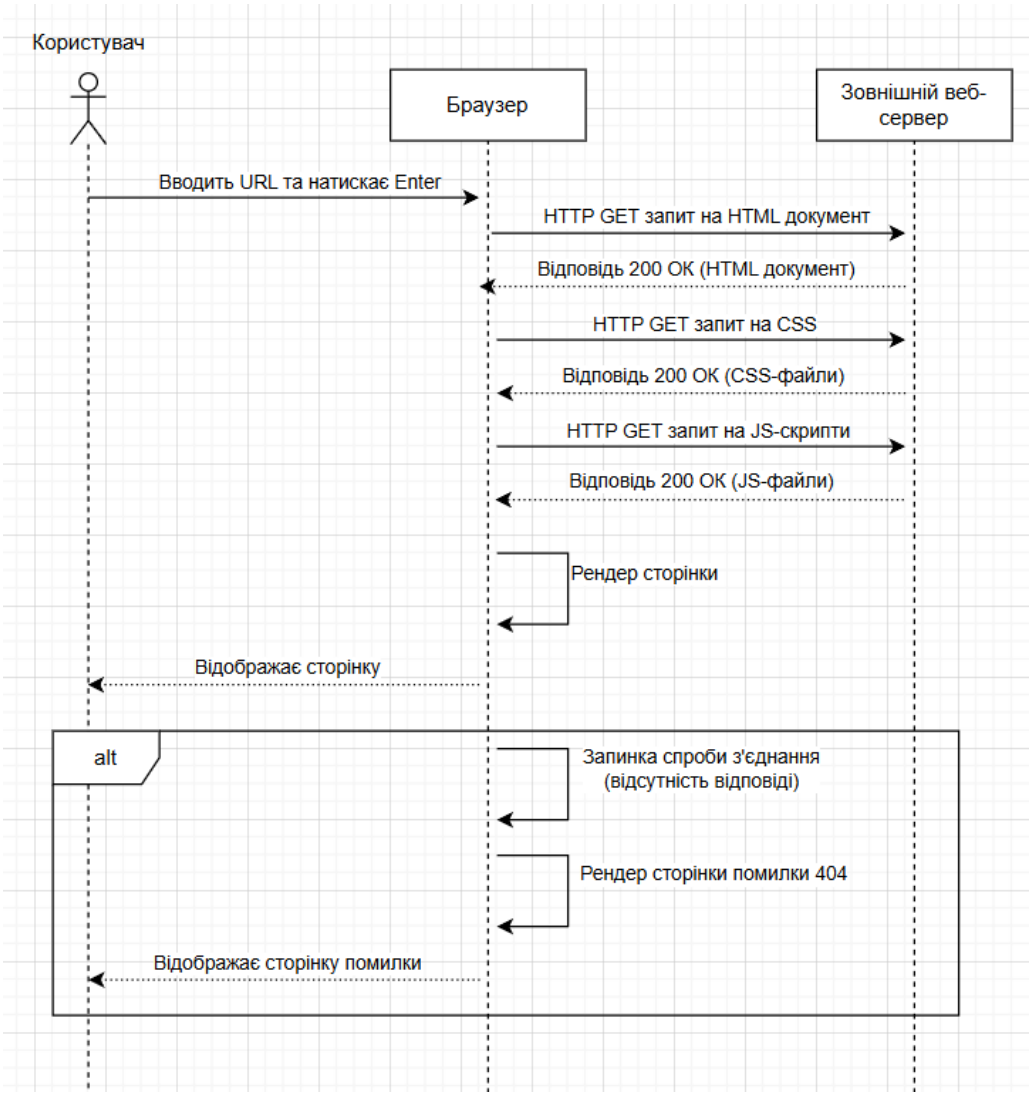

Рис.3 – Діаграма послідовності завантаження веб-сторінки

Основний перебіг подій (успішний сценарій)

- Користувач вводить URL-адресу та ініціює перехід у Браузері.
- Браузер послідовно відправляє HTTP-запити на Зовнішній веб-сервер для отримання основного HTML-документа, а потім файлів стилів та скриптів.
- Зовнішній веб-сервер на кожен запит повертає відповідний ресурс.
- Браузер отримує всі файли та виконує внутрішній процес рендерингу, об'єднуючи їх у готову веб-сторінку.
- Браузер показує повністю завантажену сторінку Користувачеві.

Альтернативний перебіг подій (сценарій помилки)

- У разі, якщо Браузеру не вдається отримати відповідь від сервера (з'єднання переривається або спливає час очікування), ініціюється сценарій помилки.
- Браузер припиняє спробу з'єднання.
- Браузер самостійно формує сторінку з інформацією про помилку.
- Браузер показує Користувачеві сторінку помилки.

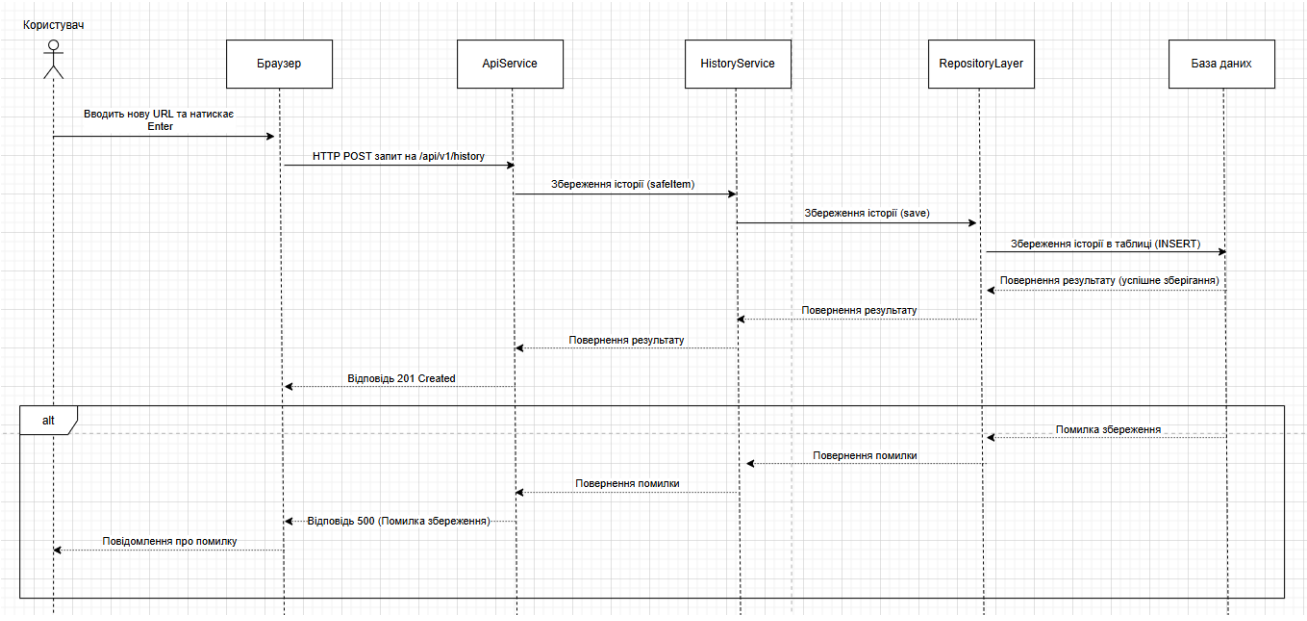

Рис.4 – Діаграма послідовності збереження історії

Основний перебіг подій (успішний сценарій)

- Процес починається автоматично після того, як Користувач переходить на новий URL у Браузері.
- Браузер надсилає фоновий HTTP POST-запит до компонента ApiService.
- ApiService делегує запит на збереження до HistoryService, який, у свою чергу, викликає відповідний метод у RepositoryLayer.

- RepositoryLayer виконує SQL-запит INSERT до Бази даних.
- Після успішного виконання операції База даних повертає підтвердження.
- Позитивний результат передається по ланцюжку назад до ApiService, який завершує операцію, надсилаючи Браузеру відповідь зі статусом 201 Created.

Альтернативний перебіг подій (сценарій помилки)

- Цей сценарій, позначений у блоці alt, активується, якщо під час взаємодії RepositoryLayer з Базою даних виникає помилка збереження.
- RepositoryLayer отримує повідомлення про помилку.
- Ця помилка передається вгору по ланцюжку викликів: від RepositoryLayer до HistoryService, а потім до ApiService.
- ApiService, обробивши виняткову ситуацію, формує HTTP-відповідь зі статусом помилки сервера (наприклад, 500 Internal Server Error) і надсилає її Браузеру.
- Браузер отримує відповідь про невдачу і повідомляє про це Користувача.

Код програми

BrowserApp.java

```
package com.example.client;

import javafx.application.Application;
import javafx.concurrent.Worker;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.HBox;
import javafx.scene.web.WebView;
import javafx.stage.Stage;

import java.net.URI;
import java.net.http.HttpClient;
```

```
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;

public class BrowserApp extends Application {

   private static final String HISTORY_API_URL = "http://localhost:8080/browser-service/api/history";
   private final HttpClient httpClient = HttpClient.newHttpClient();

   @Override
   public void start(Stage primaryStage) {
      TextField addressBar = new TextField("https://www.google.com");
      Button goButton = new Button("Go");
      WebView webView = new WebView();
      Label statusLabel = new Label("Status: Ready");

      HBox topBar = new HBox(10, new Label("URL:"), addressBar, goButton);
      topBar.setPadding(new Insets(10));

      BorderPane root = new BorderPane();
      root.setTop(topBar);
      root.setCenter(webView);
      root.setBottom(statusLabel);

      goButton.setOnAction(e -> {
         String url = addressBar.getText();
         if (!url.startsWith("http")) {
            url = "http://" + url;
         }
         statusLabel.setText("Loading: " + url);
         webView.getEngine().load(url);
      });

      webView.getEngine().getLoadWorker().stateProperty().addListener((obs, oldState, newState) -> {
```

```
        if (newState == Worker.State.SUCCEEDED) {
            String loadedUrl = webView.getEngine().getLocation();
            statusLabel.setText("Loaded: " + loadedUrl);
            saveToHistory(loadedUrl, statusLabel); // Виклик нашого бекенду
        } else if (newState == Worker.State.FAILED) {
            statusLabel.setText("Failed to load page.");
        }
    });

    Scene scene = new Scene(root, 1024, 768);
    primaryStage.setTitle("Mini Browser");
    primaryStage.setScene(scene);
    primaryStage.show();
}

private void saveToHistory(String url, Label statusLabel) {
    String jsonPayload = String.format("{\"url\":\"%s\"}", url);

    HttpRequest request = HttpRequest.newBuilder()
            .uri(URI.create(HISTORY_API_URL))
            .header("Content-Type", "application/json")
            .POST(HttpRequest.BodyPublishers.ofString(jsonPayload))
            .build();

    httpClient.sendAsync(request, HttpResponse.BodyHandlers.ofString())
            .thenAccept(response -> {
                javafx.application.Platform.runLater(() -> {
                    if (response.statusCode() == 201) {
                        statusLabel.setText("Loaded: " + url + " (History saved)");
                    } else {
                        statusLabel.setText("Loaded: " + url + " (Failed to save history: " +
response.statusCode() + ")");
                    }
                });
            })
```

```
            .exceptionally(ex -> {
                javafx.application.Platform.runLater(() -> {
                  statusLabel.setText("History service connection error.");
                });
                return null;
            });
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

HistoryItem.java

```
package com.example.model;

import java.time.LocalDateTime;

public record HistoryItem(Long id, String url, LocalDateTime visitTime) {
    public HistoryItem(String url) {
        this(null, url, null);
    }
}
```

HistoryRepository.java

```
package com.example.repository;

import com.example.model.HistoryItem;
import jakarta.enterprise.context.ApplicationScoped;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.time.LocalDateTime;
```

```
@ApplicationScoped
public class HistoryRepository {

   private static final String DB_URL =
"jdbc:postgresql://localhost:5432/browser_db";
   private static final String USER = "browserAdmin ";
   private static final String PASS = "browserAdmin ";

   public void save(HistoryItem item) throws SQLException {
      String sql = "INSERT INTO HistoryItem (url, visit_time) VALUES (?, ?)";

      try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
          PreparedStatement pstmt = conn.prepareStatement(sql)) {

         pstmt.setString(1, item.url());
         pstmt.setTimestamp(2, Timestamp.valueOf(LocalDateTime.now()));
         pstmt.executeUpdate();
      }
   }
}
```

HistoryService.java

```
package com.example.service;

import com.example.model.HistoryItem;
import com.example.repository.HistoryRepository;
import jakarta.enterprise.context.ApplicationScoped;
import jakarta.inject.Inject;
import java.sql.SQLException;

@ApplicationScoped
public class HistoryService {

   @Inject
```

```
    private HistoryRepository historyRepository;

    public void saveHistoryItem(HistoryItem item) throws SQLException {
        historyRepository.save(item);
    }
}
```

HistoryResource.java

```
package com.example.api;

import com.example.model.HistoryItem;
import com.example.service.HistoryService;
import jakarta.inject.Inject;
import jakarta.ws.rs.Consumes;
import jakarta.ws.rs.POST;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.core.MediaType;
import jakarta.ws.rs.core.Response;

@Path("/history")
public class HistoryResource {

    @Inject
    private HistoryService historyService;

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    public Response addHistory(HistoryItem item) {
        try {
            historyService.saveHistoryItem(item);
            return Response.status(Response.Status.CREATED).build();
        } catch (Exception e) {
            e.printStackTrace();
            return Response.status(Response.Status.INTERNAL_SERVER_ERROR)
                    .entity("Помилка збереження: " + e.getMessage())
```

```
                .build();
        }
    }
}
```

Контрольні запитання

1. Що собою становить діаграма розгортання?

   Діаграма розгортання (Deployment Diagram) у UML показує фізичне розміщення апаратних вузлів (серверів, клієнтів) і компонентів системи на цих вузлах. Вона відображає, як програмне забезпечення реалізується на апаратних елементах.

2. Які бувають види вузлів на діаграмі розгортання?

   Фізичні вузли (Node): реальні апаратні пристрої (сервер, комп'ютер, мобільний пристрій).

   Вузли виконання (Execution Environment): середовище, у якому запускаються компоненти (операційна система, віртуальна машина, контейнер).

3. Які бувають зв'язки на діаграмі розгортання?

   Ассоціація між вузлами (Communication Path): показує можливість обміну даними між вузлами.

   Залежність (Dependency): вказує, що один вузол або компонент залежить від іншого.

4. Які елементи присутні на діаграмі компонентів?
   - Компоненти (Component): самостійні частини ПЗ.
   - Інтерфейси (Interface): порти, через які компоненти взаємодіють.
   - Пакети (Package): групування компонентів.
   - Зв'язки (Dependency, Association): взаємозв'язки між компонентами.

5. Що становлять собою зв'язки на діаграмі компонентів?

   Зв'язки відображають залежності між компонентами: хто на кого покладається, хто надає чи використовує інтерфейс іншого компонента.

6. Які бувають види діаграм взаємодії?

Діаграма послідовностей (Sequence Diagram) – показує порядок повідомлень між об’єктами.

Діаграма комунікацій (Communication Diagram) – показує зв’язки між об’єктами та обмін повідомленнями.

Діаграма часу (Timing Diagram) – показує зміни станів об’єктів у часі.

Діаграма взаємодії (Interaction Overview Diagram) – поєднує елементи кількох діаграм взаємодії.

7. Для чого призначена діаграма послідовностей?

   Вона описує порядок і часову послідовність взаємодії між об’єктами системи для реалізації певного сценарію чи варіанту використання.

8. Які ключові елементи можуть бути на діаграмі послідовностей?

   - Об’єкти/Актори (Lifelines)
   - Повідомлення (Messages) – виклики методів між об’єктами
   - Активності (Activation bars) – час виконання дії об’єкта
   - Події створення/знищення об’єктів

9. Як діаграми послідовностей пов’язані з діаграмами варіантів використання?

   Кожна діаграма послідовностей реалізує сценарій певного варіанту використання, деталізуючи, як актори взаємодіють із системою крок за кроком.

10. Як діаграми послідовностей пов’язані з діаграмами класів?

    Повідомлення на діаграмі послідовностей зазвичай відповідають методам класів, а об’єкти на діаграмі є екземплярами класів із діаграми класів.

Висновки

Під час виконання даної лабораторної роботи я навчився проєктувати та створювати діаграми розгортання, діаграми компонентів та діаграми послідовностей, також я реалізував мінімальну робочу модель веб браузеру зі збереженням історії.