Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

**Лабораторна робота №9**

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Взаємодія компонентів системи»

Варіант: E-mail клієнт

**Виконав:**
студент групи ІА-32
Карповець Роман

Київ - 2025

**Тема:** Взаємодія компонентів системи.

**Мета:** Вивчити види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.

**Тема проєкту:** E-mail клієнт

**Опис:** Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

**Зміст**

## Теоретичні відомості

### Клієнт-серверна архітектура

Клієнт-серверні додатки являють собою найпростіший варіант розподілених додатків, де виділяється два види додатків: клієнти (представляють додаток користувачеві) і сервери (використовується для зберігання і обробки даних). Розрізняють тонкі клієнти і товсті клієнти.

**Тонкий клієнт** – клієнт, який повністю всі операції (або більшість, пов'язаних з логікою роботи програми) передає для обробки на сервер, а сам зберігає лише візуальне уявлення одержуваних від сервера відповідей. Грубо кажучи, тонкий клієнт – набір форм відображення і канал зв'язку з сервером. Прикладом тонкого клієнта є класичні Web-застосунки.

У такому варіанті використання майже все навантаження лягає на сервер або групу серверів.

Перевагою таких моделей є простота розгортання, тому що оновлювати потрібно лише сервери і в результаті клієнти з наступними запитами автоматично будуть працювати з оновленою системою.

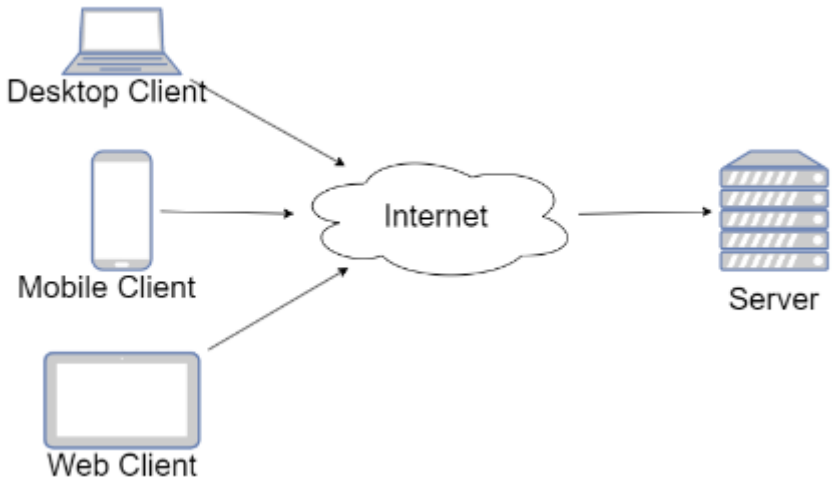


Рис. 1. Клієнт-серверна архітектура

**Товстий клієнт** – антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких

100

випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими

комп'ютерами. Перевагою такого підходу є менші вимоги до серверної частини. Також перевагою, при певному підході до реалізації, є можливість працювати клієнтам без тимчасового доступу до серверу. Прикладом товстого клієнта можна назвати мобільні застосунки, або десктоп застосунки. Наприклад, Evernote, Viber, MS Outlook, комп’ютерні антивіруси, ігри, що потребують інсталяції (The Sims, GTA, …) та інші.

Проміжним варіантом можна назвати SPA (Single Page Application) – це товсті Web-клієнти, які при старті кожен раз завантажуються з сервера, а надалі працюють з сервером через web-API. З одного боку більшу частину логіки вони відпрацьовують на клієнтський стороні, за рахунок чого зменшується серверне навантаження. Також оновлення простіше ніж для товстих клієнтів. Але такі застосунки не працюють, якщо сервер не доступний.

Клієнт-серверна взаємодія, як правило, організовується за допомогою 3-х рівневої структури: клієнтська частина, загальна частина, серверна частина. Оскільки велика частина даних загальна (класи, використовувані системою), їх прийнято виносити в загальну частину (middleware) системи. Клієнтська частина містить візуальне відображення і логіку обробки дії користувача; код для встановлення сеансу зв'язку з сервером і виконання відповідних викликів.

Серверна частина містить основну логіку роботи програми (бізнес-логіку) або ту її частину, яка відповідає зберіганню або обміну даними між клієнтом і сервером або клієнтами.

**Завдання**

• Ознайомитись з короткими теоретичними відомостями.

• Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.

• Реалізувати функціонал для роботи в розподіленому оточенні відповідно до обраної теми.

• Реалізувати взаємодію розподілених частин:

– Для клієнт-серверних варіантів: реалізація клієнтської і серверної

частини додатків, а також загальної частини (middleware); зв'язок клієнтської і серверної частин за допомогою WCF, TcpClient, .NET-Remoting на розсуд виконавця.

– Для однорангових мереж: реалізація взаємодії клієнтських додатків за допомогою WCF Peer to peer channel.

– Для SOA додатків: реалізація сервісу, що надає послуги клієнтським застосуванням; викладання сервісу в хмару або підняття у вигляді Web Service на локальній машині; використання токенів для передачі даних про автентифікації, двостороннє шифрування.

• Підготувати звіт щодо виконання лабораторної роботи. Поданий звіт повинен містити: діаграму класів, яка представляє спроєктовану архітектуру. Навести фрагменти програмного коду, які є суттєвими для відображення реалізованої архітектури

## Хід роботи

### Діаграма розгортання

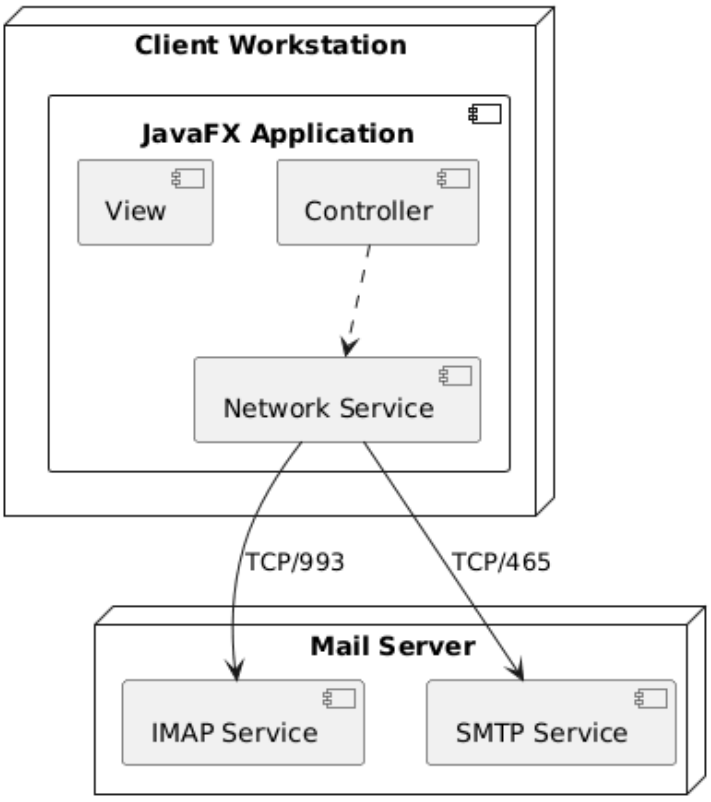


Рис. 2. Діаграма розгортання додатку

## Фрагмент коду з внесеними змінами

### EmailService.java

```
package com.emailclient.network;

import com.emailclient.service.receiver.MailReceiver;
import javafx.concurrent.Service;
import javafx.concurrent.Task;

public class EmailService extends Service<Void> {

   private final MailReceiver receiver;

   public EmailService(MailReceiver receiver) {
       this.receiver = receiver;
   }

   @Override
   protected Task<Void> createTask() {
       return new Task<>() {
           @Override
           protected Void call() throws Exception {
               Thread.sleep(1000);
               receiver.downloadEmails();
               return null;
           }
       };
   }
}
```

### MainController.java

```
package com.emailclient.view;

import com.emailclient.network.EmailService;
import com.emailclient.service.receiver.ImapReceiver;
import javafx.fxml.FXML;
import javafx.scene.control.ListView;

public class MainController {

   @FXML
   private ListView<String> emailListView;
```

```
    @FXML
    public void onCheckMailButtonClick() {
        emailListView.getItems().add("Завантаження...");

        EmailService service = new EmailService(new ImapReceiver());

        service.setOnSucceeded(event -> {
            emailListView.getItems().clear();
            emailListView.getItems().add("Лист від: example1@gmail.com | Тема: Lab 9");
            emailListView.getItems().add("Лист від: example2@gmail.com | Тема: Hello");
        });

        service.setOnFailed(event -> {
            emailListView.getItems().clear();
            emailListView.getItems().add("Помилка з'єднання!");
            service.getException().printStackTrace();
        });

        service.start();
    }
}
```

## LoginController.java

```
package com.emailclient.view;

import com.emailclient.config.SettingsManager;
import com.emailclient.database.DatabaseHandler;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.PasswordField;
import javafx.scene.control.TextField;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import java.io.IOException;

public class LoginController {

    @FXML
```

```
private TextField emailField;

@FXML
private PasswordField passwordField;

@FXML
protected void onLoginButtonClick() {
    String email = emailField.getText();
    String password = passwordField.getText();

    if (email.isEmpty() || password.isEmpty()) {
        showAlert("Помилка", "Заповніть всі поля!");
        return;
    }

    // Збереження в БД
    if (saveAccountToDB(email, password)) {

        // Після успішного запису в БД ми зберігаємо поточного користувача в глобальний стан
        SettingsManager.getInstance().setCurrentUserEmail(email);

        // (Опціонально) Вивід в консоль для перевірки, що Singleton спрацював
        System.out.println("Singleton оновлено. Поточна сесія для: " +
                SettingsManager.getInstance().getCurrentUserEmail());

        showAlert("Успіх", "Акаунт збережено! Перехід до головного вікна...");
        // Тут код відкриття Main Window

    } else {
        showAlert("Помилка", "Не вдалося зберегти дані.");
    }
}

private boolean saveAccountToDB(String email, String pass) {
    String insertSQL = "INSERT INTO accounts(email, password) VALUES(?, ?)";

    try (Connection conn = DatabaseHandler.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(insertSQL)) {

        pstmt.setString(1, email);
        pstmt.setString(2, pass);
        pstmt.executeUpdate();
        return true;
```

```
        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }
    }

    private void showAlert(String title, String content) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setContentText(content);
        alert.showAndWait();
    }

    private void openMainWindow() {
        try {
            FXMLLoader loader = new
FXMLLoader(getClass().getResource("/com/emailclient/view/main-view.fxml"));
            Parent root = loader.load();

            Stage stage = (Stage) emailField.getScene().getWindow();
            stage.setScene(new Scene(root));
            stage.setTitle("E-mail Client - Вхідні");
            stage.show();

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

**Висновок:** я вивчив види взаємодії додатків (Client-Server, Peer-to-Peer, Service-oriented Architecture), та реалізувати в проєктованій системі одну із архітектур.