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 9
З дисципліни: «Теорія розробки програмного забезпечення»
Web-browser
Client server

Виконав:
студент групи ІА-14
Калінін Я.В

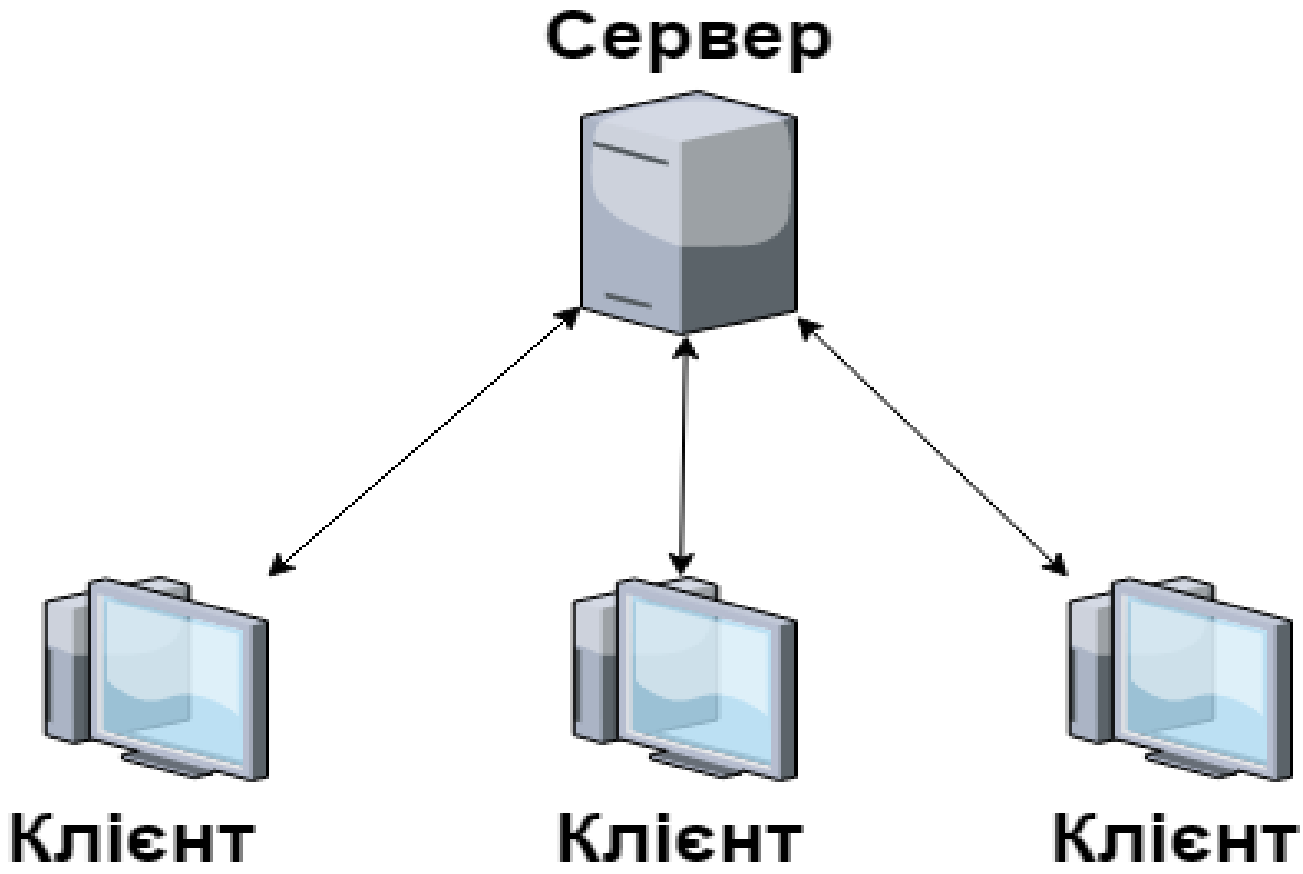
Київ 2023

Мета: Реалізувати клієнт серверну архітектуру в проєкті на тему web-browser

Клієнт-серверна архітектура - це спосіб організації програм, де взаємодіють два основні елементи: клієнт та сервер.

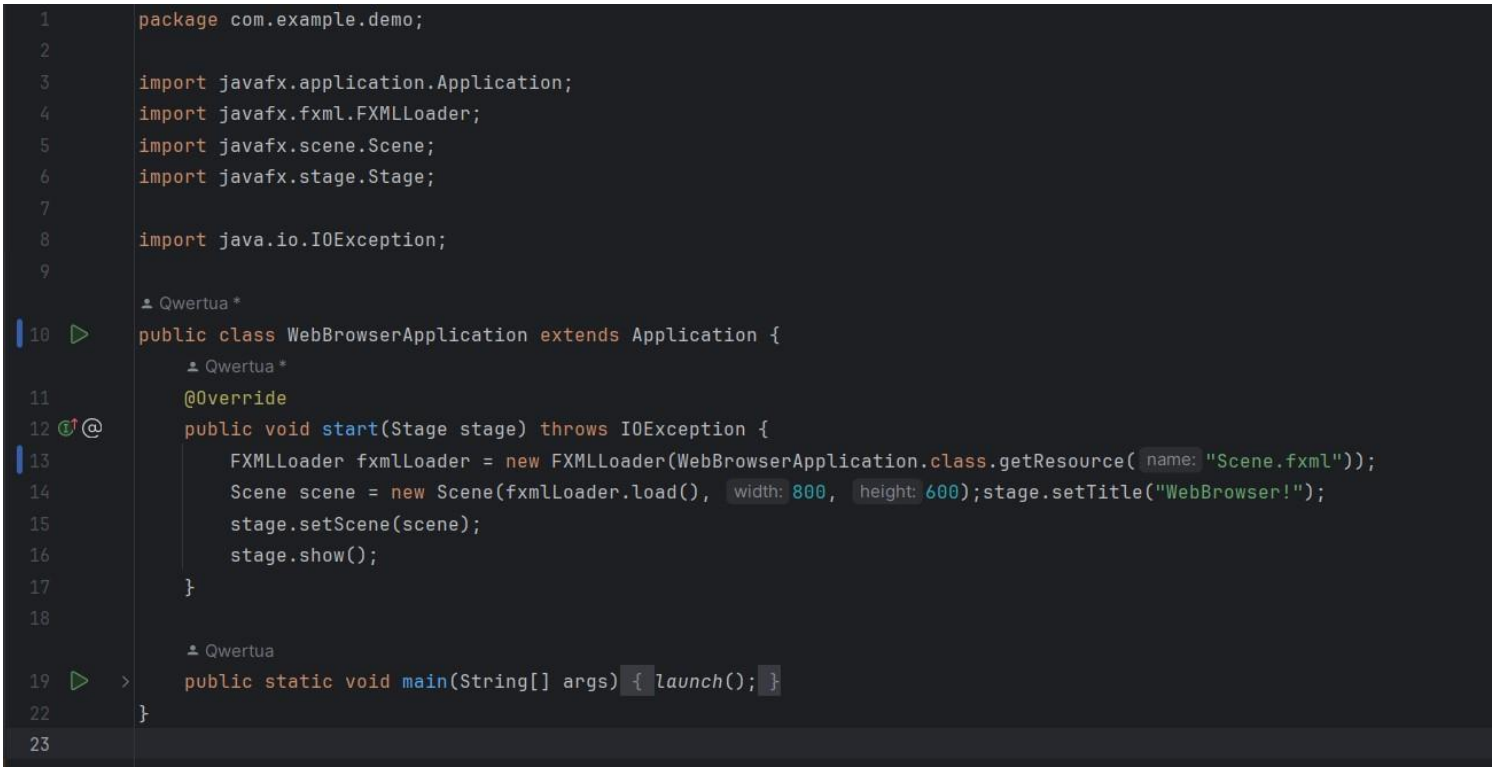
Сервер: Це програма або пристрій, який надає ресурси, дані, послуги або функції. Він "слухає" запити від клієнтів і відповідає на них.(В нашому випадку зовнішній сервер)

Клієнт: Це програма або пристрій, який відправляє запити до сервера для отримання певних ресурсів або послуг. Клієнт чекає на відповідь сервера та використовує надані ресурси або дані.(в нашому випадку web-browser)



Код:

class WebBrowserApplication



```
package com.example.demo;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
import javafx.stage.Stage;

import java.io.IOException;

Qwertua *
public class WebBrowserApplication extends Application {
    Qwertua *
    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(WebBrowserApplication.class.getResource( name: "Scene.fxml"));
        Scene scene = new Scene(fxmlLoader.load(), width: 800, height: 600);stage.setTitle("WebBrowser!");
        stage.setScene(scene);
        stage.show();
    }

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

Клас WebBrowserApplication: Це точка входу в JavaFX-додаток, яка запускає графічний інтерфейс користувача (GUI). Сам додаток функціонує як клієнт, звертаючись до веб-ресурсів.

class Controller

Клас Controller: Контролер управляє GUI і взаємодіє з веб-інженером (WebEngine), який виконує запити до веб-серверів. Коли користувач вводить URL і натискає кнопку завантаження, WebEngine звертається до веб-сервера за цією адресою, демонструючи клієнт-серверну взаємодію.

```
package com.example.demo;

import com.example.demo.factory.BadGatewayHandlerFactory;
import com.example.demo.factory.ErrorHandlerFactory;
import com.example.demo.factory.NotFoundHandlerFactory;
import com.example.demo.factory.ServiceUnavailableHandlerFactory;
import com.example.demo.loader.PageLoader;
import com.example.demo.loader.ProxyPageLoader;
import com.example.demo.loader.RealPageLoader;
import javafx.application.Platform;
import javafx.collections.ObservableList;
import javafx.concurrent.Worker;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
```

```
import javafx.scene.control.Button;
import javafx.scene.control.ProgressBar;
import javafx.scene.control.TextField;
import javafx.scene.web.WebEngine;
import javafx.scene.web.WebHistory;
import javafx.scene.web.WebView;

import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.ResourceBundle;

public class Controller implements Initializable {
    @FXML
    private Button forwardButton;
    @FXML
    private Button backButton;
    @FXML
    private ProgressBar progressBar;
    @FXML
    private WebView webView;
    @FXML
    private TextField textField;
    @FXML
    private Button displayHtmlButton;
    @FXML
    private Button executeJsButton;
    @FXML
    private Button closeButton;


    private PageLoader pageLoader;
    private WebEngine engine;
    private WebHistory history;


    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        engine = webView.getEngine();
        RealPageLoader realPageLoader = new RealPageLoader(engine);
        List<ErrorHandlerFactory> errorHandlerFactories = new ArrayList<>();
        errorHandlerFactories.add(new NotFoundHandlerFactory());
        errorHandlerFactories.add(new BadGatewayHandlerFactory());
        errorHandlerFactories.add(new ServiceUnavailableHandlerFactory());

        pageLoader = new ProxyPageLoader(realPageLoader, progressBar,
errorHandlerFactories);
        textField.setText("www.google.com");
        displayHtmlButton.setOnAction(event -> displayHtmlStructureOnPage());
        executeJsButton.setOnAction(event -> executeJavaScript());

        loadPage();
        closeButton.setDisable(true);
    }


    public void loadPage() {
        pageLoader.loadPage(textField.getText());

        pageLoader.getLoadWorkerStateProperty().addListener((observable, oldValue, newValue)
-> {
            if (newValue == Worker.State.SUCCEEDED) {
                history = engine.getHistory();
                UpdateButtonStatus();

            }
```

```
        });
    }


    public void refreshPage() {
        pageLoader.reloadPage();
    }

    public void zoomIn() {
        webView.setZoom(webView.getZoom() + 0.25);
    }

    public void zoomOut() {
        webView.setZoom(webView.getZoom() - 0.25);
    }

    public void displayHistory() {
        history = engine.getHistory();
        ObservableList<WebHistory.Entry> entries = history.getEntries();
        for (WebHistory.Entry entry : entries) {
            System.out.println(entry.getUrl() + " " + entry.getLastVisitedDate());
        }
    }

    public void goBack() {

        history = engine.getHistory();
        ObservableList<WebHistory.Entry> entries = history.getEntries();
        int index = history.getCurrentIndex();
        if (index > 0) {
            history.go(-1);
            textField.setText(entries.get(history.getCurrentIndex()).getUrl());
        }
        UpdateButtonStatus();

    }

    public void goForward() {

        history = engine.getHistory();
        ObservableList<WebHistory.Entry> entries = history.getEntries();
        int index = history.getCurrentIndex();
        if (index < entries.size() - 1) {
            history.go(1);
            textField.setText(entries.get(history.getCurrentIndex()).getUrl());
        }
        UpdateButtonStatus();

    }

    public void UpdateButtonStatus() {
        int currentIndex = history.getCurrentIndex();
        int totalEntries = history.getEntries().size();

        backButton.setDisable(currentIndex <= 0);
        forwardButton.setDisable(currentIndex >= totalEntries - 1);
    }

    /**
     * Відображає структуру HTML поточної веб-сторінки.
     * Отримує повний HTML поточної сторінки та відображає його в WebEngine.
     */
    public void displayHtmlStructureOnPage() {
        String htmlContent = (String)
engine.executeScript("document.documentElement.outerHTML");
        ContentVisitor visitor = new DisplayContentVisitor(engine);
```

```
        visitor.visitHtmlContent(htmlContent);
        closeButton.setDisable(false);
    }

 /*   private String escapeHtml(String html) {
        return html
                .replace("&", "&amp;")
                .replace("<", "&lt;")
                .replace(">", "&gt;")
                .replace("\"", "&quot;")
                .replace("'", "&#39;");
    }*/

    /**
     * Виконує JavaScript на поточній веб-сторінці та відображає вихідні дані.
     * Збирає та відображає зміст усіх тегів <script> на поточній сторінці.
     */
    public void executeJavaScript() {
        String script = """
            var scriptElements = document.getElementsByTagName('script');
            var jsContent = '';
            for (var i = 0; i < scriptElements.length; i++) {
                jsContent += scriptElements[i].outerHTML + '\\n\\n';
            }
            jsContent;
            """;
        String jsContent = (String) engine.executeScript(script);
        ContentVisitor visitor = new DisplayContentVisitor(engine);
        visitor.visitJsContent(jsContent);
        closeButton.setDisable(false);
    }
    public void closePageView() {
        closeButton.setDisable(true);
        // Додайте код для перезавантаження сторінки тут
        loadPage();


    }
}
```

Класи PageLoader і його нащадки (RealPageLoader, ProxyPageLoader): Вони відповідають за завантаження веб-сторінок. RealPageLoader безпосередньо звертається до веб-серверів отримуючи веб-контент.

```java
package com.example.demo.loader;
import ...

2 implementations  Qwertua
public interface PageLoader {
    /* Завантаження сторінки з вказаним URL*/
    2 usages  2 implementations  Qwertua
    void loadPage(String url);
    /* метод для перезавантаження сторінки*/
    2 usages  2 implementations  Qwertua
    void reloadPage();

    /*Цей метод , який відображає стан робочого потоку, використовуваного для завантаження сторінки.
    В нащому випадку дозволяє спостерігати за змінами стану робочого потоку для визначення
    завершення завантаження сторінки.
    */
    4 usages  2 implementations  Qwertua
    ReadOnlyObjectProperty<Worker.State> getLoadWorkerStateProperty();
}
```

```java
Qwertua
public class ProxyPageLoader implements PageLoader {
    10 usages
    private RealPageLoader realPageLoader;
    5 usages
    private ProgressBar progressBar;
    1 usage
    private ResponseHandler errorHandlerChain;

    1 usage  Qwertua
    public ProxyPageLoader(RealPageLoader realPageLoader, ProgressBar progressBar, List<ErrorHandlerFactory> errorHandlerFa
        this.realPageLoader = realPageLoader;
        this.progressBar = progressBar;
        this.errorHandlerChain = buildErrorHandlerChain(errorHandlerFactories);
    }
```

```java
1 usage  Qwertua
@
private ResponseHandler buildErrorHandlerChain(List<ErrorHandlerFactory> errorHandlerFactories) {
    ResponseHandler firstHandler = null;
    ResponseHandler previousHandler = null;

    for (ErrorHandlerFactory factory : errorHandlerFactories) {
        ResponseHandler currentHandler = factory.createHandler();

        if (previousHandler != null) {
            previousHandler.setNextHandler(currentHandler);
        } else {
            firstHandler = currentHandler;
        }

        previousHandler = currentHandler;
    }

    return firstHandler;
}
```

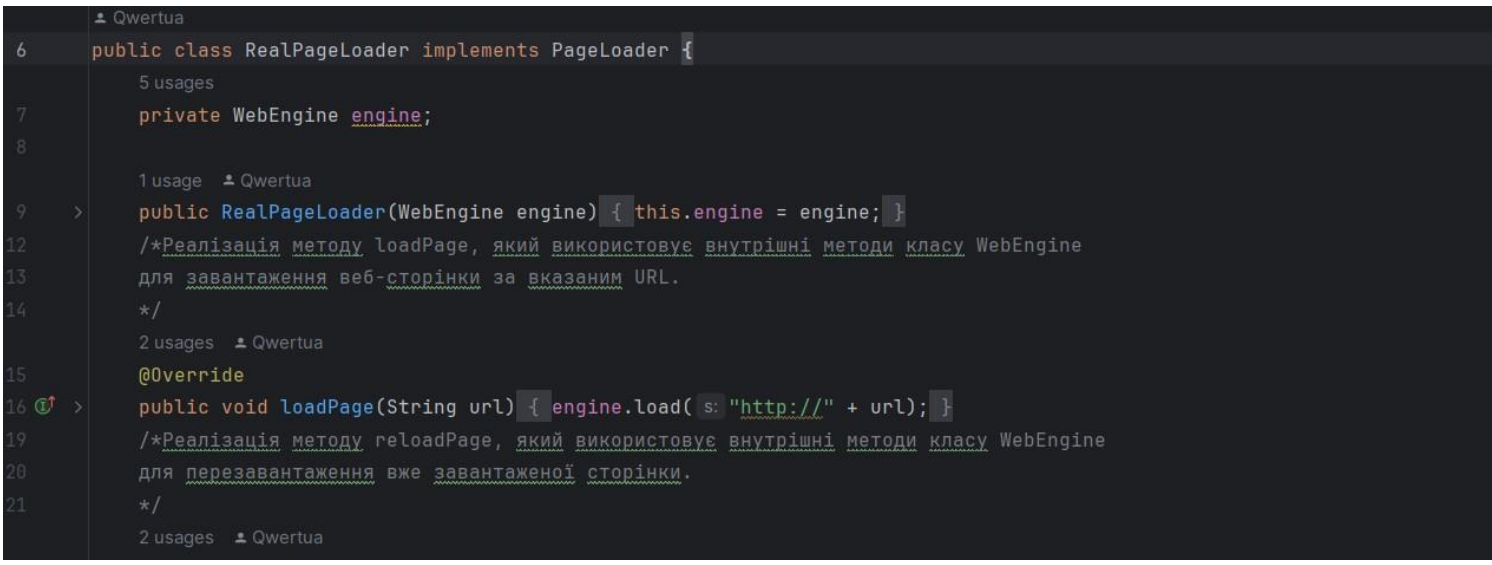
```
1 usage  Qwertua
private String generateErrorPageContent(int statusCode, String errorMessage) {
    return String.format("<html><body><h1>Error %d: %s</h1></body></html>", statusCode, errorMessage);
}

1 usage  Qwertua
private void showErrorResponsePage(int statusCode, String errorMessage) {
    Platform.runLater(() -> {
        WebEngine errorEngine = realPageLoader.getEngine();

        String content = generateErrorPageContent(statusCode, errorMessage);
        errorEngine.loadContent(content);
    });
}
```

```
57      @Override
58      public void loadPage(String url) {
59          /*Встановлюємо видимою шкалу завантаження сторінки*/
60          progressBar.setVisible(true);
61          /*завантаження сторінки  за допомогою realPageLoader*/
62          realPageLoader.loadPage(url);
63          /*відстежуємо зміну стану за допомогою слухача подій*/
64          realPageLoader.getLoadWorkerStateProperty().addListener((observable, oldValue, newValue) -> {
65              if (newValue == Worker.State.SUCCEEDED) {
66                  /*коли стан завантаження успішний прибираємо шкалу завантаження сторінки та викликаємо метод handleresponse*
67                  progressBar.setVisible(false);
68                  handleResponse(realPageLoader.getEngine().getLocation());
69              }
70          });
71      }
72
```

```
73      @Override
74      public void reloadPage() {
75          /*Встановлюємо видимою шкалу завантаження сторінки*/
76          progressBar.setVisible(true);
77          /*перезавантаження сторінки  за допомогою realPageLoader*/
78          realPageLoader.reloadPage();
79          /*відстежуємо зміну стану за допомогою слухача подій*/
80          realPageLoader.getLoadWorkerStateProperty().addListener((observable, oldValue, newValue) -> {
81              if (newValue == Worker.State.SUCCEEDED) {
82                  /*коли стан завантаження успішний прибираємо шкалу завантаження сторінки та викликаємо метод handleresponse*
83                  progressBar.setVisible(false);
84                  handleResponse(realPageLoader.getEngine().getLocation());
85              }
86          });
87      }
88
```



```
Qwertua
public class RealPageLoader implements PageLoader {
    5 usages
    private WebEngine engine;

    1 usage  Qwertua
    public RealPageLoader(WebEngine engine) { this.engine = engine; }
    /*Реалізація методу loadPage, який використовує внутрішні методи класу WebEngine
    для завантаження веб-сторінки за вказаним URL.
    */
    2 usages  Qwertua
    @Override
    public void loadPage(String url) { engine.load( s: "http://" + url); }
    /*Реалізація методу reloadPage, який використовує внутрішні методи класу WebEngine
    для перезавантаження вже завантаженої сторінки.
    */
    2 usages  Qwertua
```

```
    @Override
    public void reloadPage() { engine.reload(); }
    /*Реалізація методу getLoadWorkerStateProperty, який повертає
    властивість стану робочого потоку для відстеження стану завантаження сторінки.
    */

    4 usages  Qwertua
    @Override
    public ReadOnlyObjectProperty<Worker.State> getLoadWorkerStateProperty() {
        return engine.getLoadWorker().stateProperty();
    }
    /*отримання доступу до внутрішніх методів WebEngine*/
    4 usages  Qwertua
    public WebEngine getEngine() { return engine; }
}
```

Обробники помилок (BadGatewayHandler, NotFoundHandler, ServiceUnavailableHandler): Ці класи обробляють різні HTTP-відповіді від сервера, такі як 502 (Bad Gateway), 404 (Not Found) та 503 (Service Unavailable), що є частиною клієнт-серверної взаємодії.

```java
package com.example.demo.handler;
Qwertua
public class BadGatewayHandler extends BaseResponseHandler {
    1 usage  Qwertua
    @Override
    public boolean handleResponse(int statusCode) {
        if (statusCode == 502) {

            /*  Обробка помилки 502 */
            return true;
        } else {
            return callNextHandler(statusCode);
        }
    }
}
```

```java
package com.example.demo.handler;
Qwertua
public class NotFoundHandler extends BaseResponseHandler {
    1 usage  Qwertua
    @Override
    public boolean handleResponse(int statusCode) {
        if (statusCode == 404) {

            /*   Обробка помилки 404*/
            return true;
        } else {
            return callNextHandler(statusCode);
        }
    }
}
```

```java
package com.example.demo.handler;

Qwertua
public class ServiceUnavailableHandler extends BaseResponseHandler {
    1 usage  Qwertua
    @Override
    public boolean handleResponse(int statusCode) {
        if (statusCode == 503) {

            /* Обробка помилки 503*/
            return true;
        } else {
            return callNextHandler(statusCode);
        }
    }
}
```

Відповіді на контрольні запитання:

У чому полягають переваги і недоліки клієнт-серверної моделі?
Переваги:
1.Централізоване управління: Усі дані зосереджені на сервері, що полегшує управління, оновлення та резервне копіювання.

2.Масштабованість: Легше масштабувати, додаючи нові сервери або оновлюючи існуючі, для обробки зростаючого обсягу запитів.

3.Ефективність ресурсів: Сервери зазвичай мають більше обчислювальних ресурсів, ніж клієнтські пристрої, що дозволяє ефективніше обробляти великі обсяги даних.

4.Спрощене обслуговування: Оновлення та виправлення помилок можуть бути виконані на сервері без необхідності оновлення кожного клієнтського пристрою.

Недоліки:
1.Залежність від мережі: Клієнти залежать від підключення до сервера, що може стати проблемою при відсутності стабільного мережевого з'єднання.

2.Ненадійсність: Якщо сервер виходить з ладу, це може призвести до зупинки роботи всієї системи.

3.Обмеження пропускної здатності: Велика кількість клієнтів може перевантажити сервер, що призводить до зниження продуктивності.

Висновок: Отже, клієнт-серверна архітектура є однією з основних та широко використовуваних моделей в проектуванні програмних систем та мережевих додатків. Ця архітектурна модель передбачає розділення програмної системи на два основні компоненти: клієнтську та серверну частини, які взаємодіють між собою через мережу. Вона відображає спосіб організації та розподілу обов'язків в системі, що робить її важливою архітектурною концепцією в розробці програмного забезпечення.