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: «Різні види взаємодії додатків:client-server,
peer-to-peer, service-oriented architecture»

Варіант №18

Виконав:
студент групи ІА-24
Гуменюк К. Е.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема.
Клієнт серверна архітектура
Мета.
Метою даної лабораторної роботи є ознайомлення з різними видами взаємодії додатків, зокрема з клієнт серверною архітектурою та її практичне застосування при розробці програмного забезпечення.

Завдання.
1 . Ознайомитися з короткими теоретичними відомостями.

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

3. Застосування одного з розглянутих шаблонів при реалізації програми

Обрана тема.

..18 Shell (total commander) (state, prototype, factory method, template method, interpreter, client-server)

Оболонка повинна вміти виконувати основні дії в системі - перегляд файлів папок в файлової системі, перемикання між дисками, копіювання, видалення, переміщення об'єктів, пошук.

Короткі теоретичні відомості.

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.

У такому варіанті використання дуже велике навантаження лягає на сервер. Раніше такий варіант був доступний, оскільки обчислень було небагато, термінали були дорогими і як пра-вило йшли з урізаним обчислювальним пристроєм. У нинішній час зі зростанням обчислювальних можливостей клієнтських машин має сенс частину обчислень перекладати на клієнтські комп'ютери.

Однак дана модель має сенс ще в захищених сценаріях - коли зайві дані не можна показувати клієнтським комп'ютерам в зв'язку з небезпекою взлому. Ще однією проблемою може бути множинний доступ - щоб уникнути конфліктності даних має сенс централізація обчислень в одному місці (на сервері) для визначення послідовності операцій та уникнення пошкодження даних.

Товстий клієнт - антипод тонкого клієнта, більшість логіки обробки даних містить на стороні клієнта. Це сильно розвантажує сервер. Сервер в таких випадках зазвичай працює лише як точка доступу до деякого іншого ресурсу (наприклад, бази даних) або сполучна ланка з іншими клієнтськими комп'ютерами.

Хід роботи.

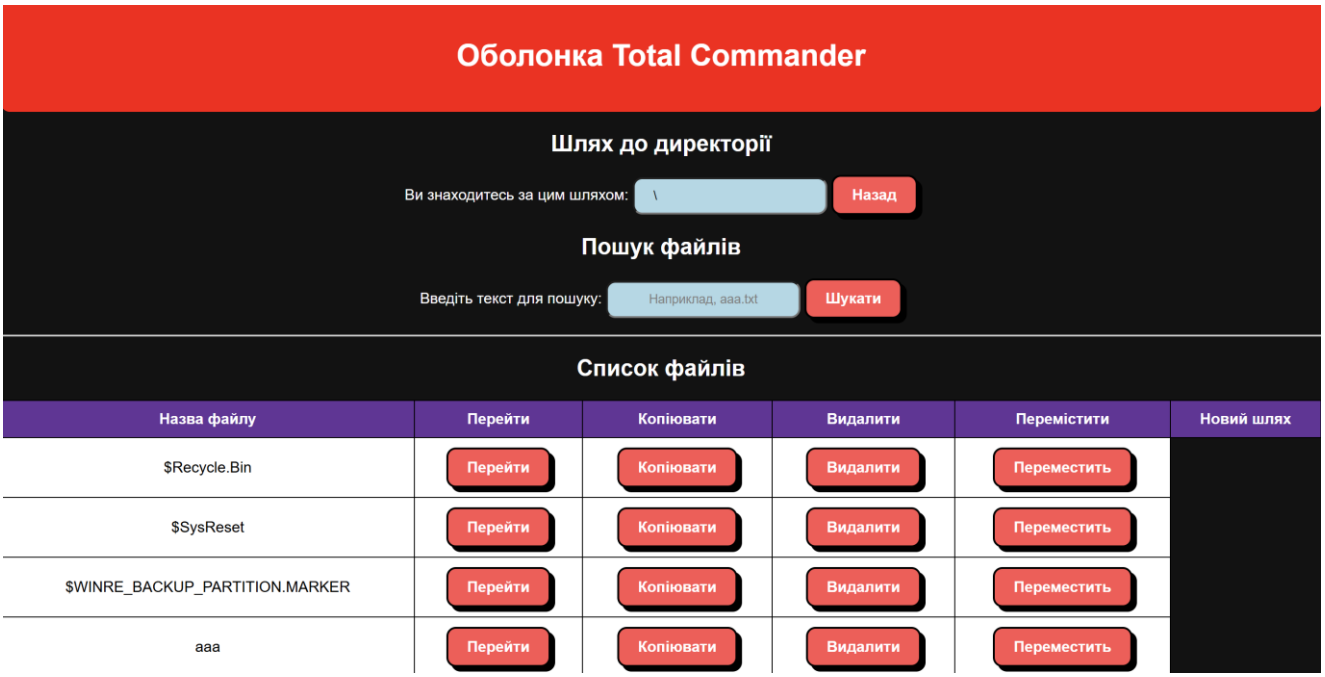

Рисунок 1.1 – Клієнтська частина програми

2025-01-24T13:20:26.780+02:00 INFO 3332 --- [shell] [nio-8080-exec-5] c.s.a.factory.ViewFileActionFactory : Initializing ViewFileAction with directory path: \

2025-01-24T13:20:26.782+02:00 INFO 3332 --- [shell] [nio-8080-exec-5] c.s.a.factory.CopyFileActionFactory : Start of action ViewFileAction

2025-01-24T13:20:26.783+02:00 INFO 3332 --- [shell] [nio-8080-exec-5] c.s.a.factory.CopyFileActionFactory : End of action ViewFileAction

2025-01-24T13:21:54.622+02:00 INFO 3332 --- [shell] [io-8080-exec-10] c.s.a.factory.ViewFileActionFactory : Initializing ViewFileAction with directory path: \aaa\

2025-01-24T13:21:54.623+02:00 INFO 3332 --- [shell] [io-8080-exec-10] c.s.a.factory.CopyFileActionFactory : Start of action ViewFileAction

2025-01-24T13:21:54.623+02:00 INFO 3332 --- [shell] [io-8080-exec-10] c.s.a.factory.CopyFileActionFactory : End of action ViewFileAction

Логи працюючої програми

Висновок.

Під час розробки ми успішно перевели наш додаток на клієнт-серверну архітектуру. Раніше це був монолітний консольний застосунок, що виконував усі команди локально. Тепер додаток розділено на:

- **Сервер:** Обробляє бізнес-логіку та виконує команди.
- **Клієнт:** Надає користувацький інтерфейс та взаємодіє з сервером через мережу.

Клієнтський застосунок, створений на Spring Boot, має консольний інтерфейс. Він зчитує введення користувача, надсилає команди на сервер і відображає результати. Інтерфейс ShellConsole спрощує взаємодію з користувачем, а також дозволяє отримувати список доступних команд із сервера, інформуючи користувача про можливості системи.

Таким чином, ми створили розподілену систему, де клієнт і сервер працюють окремо, взаємодіючи через мережу.

Додаток

```
package coursework.shell.controllers;

import coursework.shell.actions.factory.ActionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

@RestController
@RequestMapping("/actions")
public class ActionController {

    private final Map<String, ActionFactory> actionFactories;

    @Autowired
    public ActionController(Map<String, ActionFactory>
actionFactories) {
        this.actionFactories = actionFactories;
    }

    @GetMapping
    public ResponseEntity<Set<String>> getAvailableActions() {
        Set<String> actions = new
HashSet<>(actionFactories.keySet());
        return ResponseEntity.ok(actions);
    }
}
package coursework.shell.controllers;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class BaseController {

    @GetMapping("/")
    public String index() {
        return "index.html";
    }

}
package coursework.shell.controllers;

import coursework.shell.ActionFlowController;
import coursework.shell.actions.FileAction;
import coursework.shell.actions.SearchFileAction;
import coursework.shell.actions.factory.ActionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/actions")
public class SearchActionController {

    private final ActionFlowController actionFlowController;
    private final ActionFactory searchActionFactory;

    @Autowired
    public SearchActionController(ActionFlowController
actionFlowController, Map<String, ActionFactory> actionFactories) {
        this.actionFlowController = actionFlowController;
        this.searchActionFactory = actionFactories.get("search");
    }

    @PostMapping("/search")
    public ResponseEntity<?> executeSearchCommand(@RequestParam
String query, @RequestParam String path) {
        String[] args = {"search", query, path};
        FileAction fileAction =
searchActionFactory.createAction(args);

        if (fileAction == null) {
            return ResponseEntity.badRequest().body("Search
error.");
        }

        if (fileAction instanceof SearchFileAction searchFileAction)
{
            searchFileAction.setPath(path);
            actionFlowController.handleAction(fileAction);
            List<String> results = searchFileAction.getResults();
            return ResponseEntity.ok(results);
        } else {
            return ResponseEntity.status(500).body("Server error.");
        }
    }
}
package coursework.shell.controllers;

import coursework.shell.ActionFlowController;
import coursework.shell.actions.factory.ActionFactory;
import coursework.shell.actions.interpreter.Condition;
import coursework.shell.actions.interpreter.ConditionAnalyzer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.Map;
```

```
@RestController
@RequestMapping("/command-line")
public class UserOperationController {

    private final ConditionAnalyzer parser;

    @Autowired
    public UserOperationController(ActionFlowController
actionFlowController, Map<String, ActionFactory> actionFactories) {
        this.parser = new ConditionAnalyzer(actionFlowController,
actionFactories);
    }

    @PostMapping("/execute")
    public String executeCommandLine(@RequestBody String
commandLine) {
        if (commandLine == null || commandLine.trim().isEmpty()) {
            return "Command is empty";
        }

        try {
            Condition condition = parser.analizeAction(commandLine);
            condition.interpret();
            return "Success";
        } catch (Exception e) {
            return "Error: " + e.getMessage();
        }
    }
}
package coursework.shell.controllers;


import coursework.shell.ActionFlowController;
import coursework.shell.actions.FileAction;
import coursework.shell.actions.ViewFileAction;
import coursework.shell.actions.factory.ActionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/actions")
public class ViewActionController {

    private final ActionFlowController actionFlowController;
    private final ActionFactory viewActionFactory;

    @Autowired
    public ViewActionController(ActionFlowController
actionFlowController, Map<String, ActionFactory> actionFactories) {
        this.actionFlowController = actionFlowController;
        this.viewActionFactory = actionFactories.get("view");
    }
```

```
    @PostMapping("/view")
    public ResponseEntity<?> executeViewAction(@RequestParam(value =
"path", required = false) String path) {
        String[] args;
        if (path == null || path.trim().isEmpty()) {
            args = new String[]{"view"};
        } else {
            args = new String[]{"view", path};
        }
        FileAction fileAction =
viewActionFactory.createAction(args);


        if (fileAction instanceof ViewFileAction viewAction) {
            actionFlowController.handleAction(fileAction);
            List<String> fileList = viewAction.getFileList();
            return ResponseEntity.ok(fileList);
        } else {
            return ResponseEntity.status(500).body("Server error");
        }
    }
}
```