SmartEdit Text Editor

Complete Implementation Guide

Design Patterns with Full File Structure

CS434 Software Design Patterns

Özye■in University - Fall 2025

Team Members:

Mir Bedirhan KAYGUSUZ - S028260
Özgür Tuna Yavuz - S024224

1. Complete Project Structure

1.1 Overall Directory Structure

```
smartedit/
■■■ backend/ # Java Spring Boot Backend
■ ■■■ src/
■ ■ ■■■ main/
■ ■ ■■■ java/
■ ■ ■ ■■■ com/
■ ■ ■ ■■■ smartedit/
■ ■ ■ ■■■ SmartEditApplication.java
■ ■ ■ ■■■ controller/ # REST Controllers
■ ■ ■ ■■■ service/ # Business Logic
■ ■ ■ ■■■ model/ # Domain Models
■ ■ ■ ■■■ repository/ # Data Access
■ ■ ■ ■■■ patterns/ # Design Patterns
■ ■ ■ ■■■ command/ # Command Pattern
■ ■ ■ ■■■ memento/ # Memento Pattern
■ ■ ■ ■■■ strategy/ # Strategy Pattern
■ ■ ■ ■■■ observer/ # Observer Pattern
■ ■ ■ ■■■ singleton/ # Singleton Pattern
■ ■ ■ ■■■ factory/ # Factory Pattern
■ ■ ■ ■■■ decorator/ # Decorator Pattern
■ ■ ■■■ resources/
■ ■ ■■■ application.properties
■ ■■■ pom.xml # Maven dependencies
■ ■■■ README.md
■
■■■ frontend/ # React Frontend
■■■ src/
■ ■■■ components/ # React Components
■ ■ ■■■ Header.jsx
■ ■ ■■■ Sidebar.jsx
■ ■ ■■■ Editor.jsx
■ ■ ■■■ Toolbar.jsx
■ ■ ■■■ StatusBar.jsx
■ ■ ■■■ modals/
■ ■ ■■■ FindReplace.jsx
■ ■ ■■■ SaveAs.jsx
■ ■ ■■■ RestorePoints.jsx
■ ■■■ services/ # API Communication
■ ■ ■■■ editorService.js
■ ■ ■■■ fileService.js
■ ■ ■■■ commandService.js
■ ■■■ App.jsx
■ ■■■ index.js
■■■ package.json
■■■ README.md
```

1.2 Backend vs Frontend Responsibilities

Layer	Responsibilities	Technologies

Backend (Java)	• All design pattern implementations • Business logic • Data persistence • File operations • API endpoints	• Java 17+ • Spring Boot • Spring Web • Maven
Frontend (React)	• User interface • User interactions • API calls • State management (UI only) • No business logic	• React 18 • Axios (API calls) • React Hooks • CSS/Styling

2. Command Pattern Implementation

2.1 Backend Implementation

File Structure:

```
backend/src/main/java/com/smartedit/patterns/command/
■■■ Command.java # Interface
■■■ InsertTextCommand.java # Concrete Command
■■■ DeleteTextCommand.java # Concrete Command
■■■ ReplaceTextCommand.java # Concrete Command
■■■ CommandManager.java # Invoker (manages history)
■■■ CommandHistory.java # Helper class for undo/redo stack
```

2.1.1 Command Interface (Command.java)

```java
package com.smartedit.patterns.command;

public interface Command {
    void execute();
    void undo();
    String getDescription();
}
```

2.1.2 InsertTextCommand.java

```java
package com.smartedit.patterns.command;

import com.smartedit.model.Document;

public class InsertTextCommand implements Command {
    private Document document;
    private String text;
    private int position;

    public InsertTextCommand(Document doc, String text, int position) {
        this.document = doc;
        this.text = text;
        this.position = position;
    }

    @Override
    public void execute() {
        document.insert(position, text);
    }

    @Override
    public void undo() {
        document.delete(position, text.length());
    }

    @Override
    public String getDescription() {
        return "Insert: " + text;
    }
}
```

2.1.3 DeleteTextCommand.java

```
package com.smartedit.patterns.command;

import com.smartedit.model.Document;

public class DeleteTextCommand implements Command {
    private Document document;
    private int startPos;
    private int length;
    private String deletedText;  // Store for undo

    public DeleteTextCommand(Document doc, int start, int length) {
        this.document = doc;
        this.startPos = start;
        this.length = length;
    }

    @Override
    public void execute() {
        deletedText = document.getText(startPos, length);
        document.delete(startPos, length);
    }

    @Override
    public void undo() {
        document.insert(startPos, deletedText);
    }

    @Override
    public String getDescription() {
        return "Delete: " + deletedText;
    }
}
```

2.1.4 CommandManager.java

```
package com.smartedit.patterns.command;

import java.util.Stack;

public class CommandManager {
    private Stack<Command> undoStack = new Stack<>();
    private Stack<Command> redoStack = new Stack<>();

    public void executeCommand(Command command) {
        command.execute();
        undoStack.push(command);
        redoStack.clear();  // Clear redo stack on new command
    }

    public void undo() {
        if (!undoStack.isEmpty()) {
            Command command = undoStack.pop();
            command.undo();
            redoStack.push(command);
        }
    }

    public void redo() {
        if (!redoStack.isEmpty()) {
            Command command = redoStack.pop();
            command.execute();
            undoStack.push(command);
        }
    }

    public boolean canUndo() {
        return !undoStack.isEmpty();
    }

    public boolean canRedo() {
        return !redoStack.isEmpty();
    }
}
```

2.1.5 REST Controller (EditorController.java)

```
package com.smartedit.controller;

import com.smartedit.patterns.command.*;
import com.smartedit.service.EditorService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/editor")
@CrossOrigin(origins = "http://localhost:3000")
public class EditorController {

    private final EditorService editorService;

    @PostMapping("/insert")
    public ResponseEntity<EditorResponse> insertText(
            @RequestBody InsertRequest request) {

        Command cmd = new InsertTextCommand(
            editorService.getCurrentDocument(),
            request.getText(),
            request.getPosition()
        );

        editorService.getCommandManager().executeCommand(cmd);

        return ResponseEntity.ok(new EditorResponse(
            editorService.getCurrentDocument().getContent(),
            true
        ));
    }

    @PostMapping("/undo")
    public ResponseEntity<EditorResponse> undo() {
        editorService.getCommandManager().undo();
        return ResponseEntity.ok(new EditorResponse(
            editorService.getCurrentDocument().getContent(),
            editorService.getCommandManager().canUndo()
        ));
    }

    @PostMapping("/redo")
    public ResponseEntity<EditorResponse> redo() {
        editorService.getCommandManager().redo();
        return ResponseEntity.ok(new EditorResponse(
            editorService.getCurrentDocument().getContent(),
            editorService.getCommandManager().canRedo()
        ));
    }
}
```

2.2 Frontend Implementation

File Structure:

```
frontend/src/
■■■ services/
■ ■■■ commandService.js # API calls for commands
■■■ components/
■■■ Editor.jsx # Main editor component
■■■ Toolbar.jsx # Undo/Redo buttons
```

2.2.1 commandService.js

```
// frontend/src/services/commandService.js
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/editor';

export const commandService = {
    insertText: async (text, position) => {
        const response = await axios.post(`${API_URL}/insert`, {
            text: text,
            position: position
        });
        return response.data;
    },

    deleteText: async (startPos, length) => {
        const response = await axios.post(`${API_URL}/delete`, {
            startPos: startPos,
            length: length
        });
        return response.data;
    },

    undo: async () => {
        const response = await axios.post(`${API_URL}/undo`);
        return response.data;
    },

    redo: async () => {
        const response = await axios.post(`${API_URL}/redo`);
        return response.data;
    }
};
```

2.2.2 Editor.jsx (with Command Pattern)

```
// frontend/src/components/Editor.jsx
import React, { useState } from 'react';
import { commandService } from '../services/commandService';

const Editor = () => {
    const [content, setContent] = useState('');
    const [canUndo, setCanUndo] = useState(false);
    const [canRedo, setCanRedo] = useState(false);

    const handleTextChange = async (e) => {
        const newText = e.target.value;
        const cursorPos = e.target.selectionStart;

        // Call backend to execute command
        const response = await commandService.insertText(
            newText.slice(cursorPos - 1, cursorPos),
            cursorPos - 1
        );

        setContent(response.content);
        setCanUndo(response.canUndo);
    };

    const handleUndo = async () => {
        const response = await commandService.undo();
        setContent(response.content);
        setCanUndo(response.canUndo);
        setCanRedo(response.canRedo);
    };

    const handleRedo = async () => {
        const response = await commandService.redo();
        setContent(response.content);
        setCanUndo(response.canUndo);
        setCanRedo(response.canRedo);
    };

    return (
        <div>
            <button onClick={handleUndo} disabled={!canUndo}>
                Undo
            </button>
            <button onClick={handleRedo} disabled={!canRedo}>
                Redo
            </button>
            <textarea
                value={content}
                onChange={handleTextChange}
            />
        </div>
    );
};
```

3. Memento Pattern Implementation

3.1 Backend Implementation

File Structure:

```
backend/src/main/java/com/smartedit/patterns/memento/
■■■ DocumentMemento.java # Memento class (snapshot)
■■■ Document.java # Originator
■■■ MementoManager.java # Caretaker
■■■ SnapshotInfo.java # Metadata for snapshots
```

3.1.1 DocumentMemento.java

```java
package com.smartedit.patterns.memento;

import java.time.LocalDateTime;

public class DocumentMemento {
    private final String content;
    private final int cursorPosition;
    private final LocalDateTime timestamp;
    private final String fileName;

    public DocumentMemento(String content, int cursor, String fileName) {
        this.content = content;
        this.cursorPosition = cursor;
        this.timestamp = LocalDateTime.now();
        this.fileName = fileName;
    }

    public String getContent() { return content; }
    public int getCursorPosition() { return cursorPosition; }
    public LocalDateTime getTimestamp() { return timestamp; }
    public String getFileName() { return fileName; }
}
```

3.1.2 Document.java (with Memento methods)

```java
package com.smartedit.model;

import com.smartedit.patterns.memento.DocumentMemento;

public class Document {
    private String content;
    private int cursorPosition;
    private String fileName;

    // Create memento (snapshot)
    public DocumentMemento createMemento() {
        return new DocumentMemento(
            content,
            cursorPosition,
            fileName
        );
    }

    // Restore from memento
    public void restore(DocumentMemento memento) {
        this.content = memento.getContent();
        this.cursorPosition = memento.getCursorPosition();
        this.fileName = memento.getFileName();
    }

    // Getters and setters
    public String getContent() { return content; }
    public void setContent(String content) { this.content = content; }
}
```

3.1.3 MementoManager.java (Caretaker)

```java
package com.smartedit.patterns.memento;

import java.util.*;

public class MementoManager {
    private Map<String, DocumentMemento> snapshots = new HashMap<>();

    public String saveSnapshot(DocumentMemento memento) {
        String id = UUID.randomUUID().toString();
        snapshots.put(id, memento);
        return id;
    }

    public DocumentMemento getSnapshot(String id) {
        return snapshots.get(id);
    }

    public List<SnapshotInfo> getAllSnapshots() {
        List<SnapshotInfo> list = new ArrayList<>();
        for (Map.Entry<String, DocumentMemento> entry : snapshots.entrySet()) {
            DocumentMemento m = entry.getValue();
            list.add(new SnapshotInfo(
                entry.getKey(),
                m.getTimestamp(),
                m.getFileName(),
                m.getContent().substring(0, Math.min(50, m.getContent().length()))
            ));
        }
        return list;
    }

    public void deleteSnapshot(String id) {
        snapshots.remove(id);
    }
}
```

3.1.4 REST Controller for Snapshots

```java
package com.smartedit.controller;

import com.smartedit.patterns.memento.*;
import com.smartedit.service.EditorService;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/snapshot")
@CrossOrigin(origins = "http://localhost:3000")
public class SnapshotController {

    private final EditorService editorService;

    @PostMapping("/create")
    public ResponseEntity<String> createSnapshot() {
        Document doc = editorService.getCurrentDocument();
        DocumentMemento memento = doc.createMemento();
        String snapshotId = editorService.getMementoManager()
                                         .saveSnapshot(memento);
        return ResponseEntity.ok(snapshotId);
    }
    @GetMapping("/list")
    public ResponseEntity<List<SnapshotInfo>> listSnapshots() {
        List<SnapshotInfo> snapshots = editorService.getMementoManager()
                                                    .getAllSnapshots();
        return ResponseEntity.ok(snapshots);
    }

    @PostMapping("/restore/{id}")
    public ResponseEntity<EditorResponse> restoreSnapshot(
            @PathVariable String id) {
        DocumentMemento memento = editorService.getMementoManager()
                                              .getSnapshot(id);
        editorService.getCurrentDocument().restore(memento);
        return ResponseEntity.ok(new EditorResponse(
            editorService.getCurrentDocument().getContent(),
            true
        ));
    }
}
```

3.2 Frontend Implementation

File Structure:

```
frontend/src/
■■■ services/
■ ■■■ snapshotService.js # API calls for snapshots
■■■ components/
■■■ modals/
■■■ RestorePoints.jsx # Modal to show snapshots
```

3.2.1 snapshotService.js

```
// frontend/src/services/snapshotService.js
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/snapshot';

export const snapshotService = {
    createSnapshot: async () => {
        const response = await axios.post(`${API_URL}/create`);
        return response.data;
    },

    listSnapshots: async () => {
        const response = await axios.get(`${API_URL}/list`);
        return response.data;
    },

    restoreSnapshot: async (snapshotId) => {
        const response = await axios.post(
            `${API_URL}/restore/${snapshotId}`
        );
        return response.data;
    }
};
```

3.2.2 Auto-save Implementation in Editor.jsx

```
// frontend/src/components/Editor.jsx
import React, { useState, useEffect } from 'react';
import { snapshotService } from '../services/snapshotService';

const Editor = () => {
    const [content, setContent] = useState('');

    // Auto-save every 30 seconds
    useEffect(() => {
        const interval = setInterval(async () => {
            if (content) {
                const snapshotId = await snapshotService.createSnapshot();
                console.log('Auto-saved:', snapshotId);
            }
        }, 30000);  // 30 seconds

        return () => clearInterval(interval);
    }, [content]);
    return (
        <textarea
            value={content}
            onChange={(e) => setContent(e.target.value)}
        />
    );
};
```

3.2.3 RestorePoints Modal Component

```
// frontend/src/components/modals/RestorePoints.jsx
import React, { useState, useEffect } from 'react';
import { snapshotService } from '../../services/snapshotService';

const RestorePoints = ({ onClose, onRestore }) => {
    const [snapshots, setSnapshots] = useState([]);

    useEffect(() => {
        loadSnapshots();
    }, []);

    const loadSnapshots = async () => {
        const data = await snapshotService.listSnapshots();
        setSnapshots(data);
    };

    const handleRestore = async (snapshotId) => {
        const response = await snapshotService.restoreSnapshot(snapshotId);
        onRestore(response.content);
        onClose();
    };

    return (
        <div className="modal">
            <h2>Restore Points</h2>
            <div className="snapshot-list">
                {snapshots.map(snapshot => (
                    <div key={snapshot.id} className="snapshot-item">
                        <span>{snapshot.timestamp}</span>
                        <span>{snapshot.fileName}</span>
                        <p>{snapshot.preview}...</p>
                        <button onClick={() => handleRestore(snapshot.id)}>
                            Restore
                        </button>
                    </div>
                ))}
            </div>
            <button onClick={onClose}>Cancel</button>
        </div>
    );
};
```

4. Strategy Pattern Implementation

4.1 Backend Implementation

File Structure:

```
backend/src/main/java/com/smartedit/patterns/strategy/
■■■ FileSaveStrategy.java # Strategy interface
■■■ TextSaveStrategy.java # Concrete strategy for .txt
■■■ MarkdownSaveStrategy.java # Concrete strategy for .md
■■■ HTMLSaveStrategy.java # Concrete strategy for .html
■■■ FileManager.java # Context class
```

4.1.1 FileSaveStrategy.java (Interface)

```
package com.smartedit.patterns.strategy;

public interface FileSaveStrategy {
    void save(String fileName, String content);
    String getFileExtension();
}
```

4.1.2 TextSaveStrategy.java

```
package com.smartedit.patterns.strategy;

import java.io.FileWriter;
import java.io.IOException;

public class TextSaveStrategy implements FileSaveStrategy {

    @Override
    public void save(String fileName, String content) {
        try (FileWriter writer = new FileWriter(fileName + ".txt")) {
            writer.write(content);  // Save as-is, no conversion
        } catch (IOException e) {
            throw new RuntimeException("Failed to save file", e);
        }
    }

    @Override
    public String getFileExtension() {
        return "txt";
    }
}
```

4.1.3 MarkdownSaveStrategy.java

```
package com.smartedit.patterns.strategy;

import java.io.FileWriter;
import java.io.IOException;

public class MarkdownSaveStrategy implements FileSaveStrategy {
```

```java
    @Override
    public void save(String fileName, String content) {
        // Convert formatting to Markdown syntax
        String markdown = convertToMarkdown(content);

        try (FileWriter writer = new FileWriter(fileName + ".md")) {
            writer.write(markdown);
        } catch (IOException e) {
            throw new RuntimeException("Failed to save file", e);
        }
    }

    private String convertToMarkdown(String content) {
        // Convert bold: <b>text</b> to **text**
        content = content.replaceAll("<b>(.*?)</b>", "**$1**");

        // Convert italic: <i>text</i> to *text*
        content = content.replaceAll("<i>(.*?)</i>", "*$1*");

        return content;
    }

    @Override
    public String getFileExtension() {
        return "md";
    }
}
```

4.1.4 HTMLSaveStrategy.java

```java
package com.smartedit.patterns.strategy;

import java.io.FileWriter;
import java.io.IOException;

public class HTMLSaveStrategy implements FileSaveStrategy {

    @Override
    public void save(String fileName, String content) {
        String html = convertToHTML(content);

        try (FileWriter writer = new FileWriter(fileName + ".html")) {
            writer.write(html);
        } catch (IOException e) {
            throw new RuntimeException("Failed to save file", e);
        }
    }

    private String convertToHTML(String content) {
        StringBuilder html = new StringBuilder();
        html.append("<!DOCTYPE html>\n");
        html.append("<html>\n<head>\n");
        html.append("<meta charset='UTF-8'>\n");
        html.append("<title>Document</title>\n");
        html.append("</head>\n<body>\n");

        // Convert newlines to <br>
        content = content.replace("\n", "<br>\n");

        html.append(content);
        html.append("\n</body>\n</html>");

        return html.toString();
    }

    @Override
    public String getFileExtension() {
        return "html";
    }
}
```

4.1.5 FileManager.java (Context)

```java
package com.smartedit.patterns.strategy;

public class FileManager {
    private FileSaveStrategy strategy;

    public void setStrategy(FileSaveStrategy strategy) {
        this.strategy = strategy;
    }

    public void saveFile(String fileName, String content) {
        if (strategy == null) {
            throw new IllegalStateException("Strategy not set");
        }
        strategy.save(fileName, content);
    }
    public String getFileExtension() {
        return strategy != null ? strategy.getFileExtension() : "";
    }
}
```

4.1.6 REST Controller for File Operations

```
package com.smartedit.controller;

import com.smartedit.patterns.strategy.*;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/file")
@CrossOrigin(origins = "http://localhost:3000")
public class FileController {

    private final FileManager fileManager;

    @PostMapping("/save")
    public ResponseEntity<String> saveFile(@RequestBody SaveRequest request) {
        // Select strategy based on format
        FileSaveStrategy strategy;

        switch (request.getFormat()) {
            case "txt":
                strategy = new TextSaveStrategy();
                break;
            case "md":
                strategy = new MarkdownSaveStrategy();
                break;
            case "html":
                strategy = new HTMLSaveStrategy();
                break;
            default:
                return ResponseEntity.badRequest()
                    .body("Invalid format: " + request.getFormat());
        }

        fileManager.setStrategy(strategy);
        fileManager.saveFile(request.getFileName(), request.getContent());

        return ResponseEntity.ok("File saved successfully");
    }
}

// DTO classes
class SaveRequest {
    private String fileName;
    private String format;
    private String content;

    // Getters and setters
}
```

4.2 Frontend Implementation

File Structure:

```
frontend/src/
■■■ services/
■ ■■■ fileService.js # API calls for file operations
■■■ components/
■■■ modals/
■■■ SaveAs.jsx # Save As modal with format selection
```

4.2.1 fileService.js

```
// frontend/src/services/fileService.js
import axios from 'axios';

const API_URL = 'http://localhost:8080/api/file';

export const fileService = {
    saveFile: async (fileName, format, content) => {
        const response = await axios.post(`${API_URL}/save`, {
            fileName: fileName,
            format: format,      // 'txt', 'md', or 'html'
            content: content
        });
        return response.data;
    },

    loadFile: async (fileName) => {
        const response = await axios.get(`${API_URL}/load/${fileName}`);
        return response.data;
    }
};
```

4.2.2 SaveAs Modal Component

```
// frontend/src/components/modals/SaveAs.jsx
import React, { useState } from 'react';
import { fileService } from '../../services/fileService';

const SaveAs = ({ content, onClose, onSaved }) => {
    const [fileName, setFileName] = useState('untitled');
    const [format, setFormat] = useState('txt');

    const handleSave = async () => {
        try {
            await fileService.saveFile(fileName, format, content);
            onSaved(`${fileName}.${format}`);
            onClose();
        } catch (error) {
            alert('Failed to save file: ' + error.message);
        }
    };

    return (
        <div className="modal">
            <h2>Save As</h2>

            <div>
                <label>File Name:</label>
                <input
                    type="text"
                    value={fileName}
                    onChange={(e) => setFileName(e.target.value)}
                />
            </div>

            <div>
                <label>Format:</label>
                <select
                    value={format}
                    onChange={(e) => setFormat(e.target.value)}
                >
                    <option value="txt">Plain Text (.txt)</option>
                    <option value="md">Markdown (.md)</option>
                    <option value="html">HTML (.html)</option>
                </select>
            </div>

            <div>
                <button onClick={onClose}>Cancel</button>
                <button onClick={handleSave}>Save</button>
            </div>
        </div>
    );
};
```

5. Complete API Reference

Endpoint	Method	Request Body	Response	Pattern
/api/editor/insert	POST	{ text, position }	{ content, canUndo }	Command
/api/editor/delete	POST	{ startPos, length }	{ content, canUndo }	Command
/api/editor/undo	POST	-	{ content, canUndo, canRedo }	Command
/api/editor/redo	POST	-	{ content, canUndo, canRedo }	Command
/api/snapshot/create	POST	-	snapshotId (string)	Memento
/api/snapshot/list	GET	-	[{ id, timestamp, preview }]	Memento
/api/snapshot/restore/{id}	POST	-	{ content }	Memento
/api/file/save	POST	{ fileName, format, content }	success message	Strategy
/api/file/load/{name}	GET	-	{ content, format }	Strategy

6. Additional Patterns (Brief Overview)

6.1 Observer Pattern

```
Backend Location: backend/src/main/java/com/smartedit/patterns/observer/
Files:
- DocumentObserver.java (interface)
- StatusBarObserver.java
- AutoSaveObserver.java

Usage: When Document.setContent() is called, all observers are notified.
Observers calculate word count, char count, and trigger auto-save.

Frontend: Receives updated stats in API response, updates StatusBar component.
```

6.2 Singleton Pattern

```
Backend Location: backend/src/main/java/com/smartedit/patterns/singleton/
Files:
- EditorManager.java (singleton instance)

Usage:
- EditorManager.getInstance() returns single instance
- Manages currentDocument, commandManager, mementoManager
- Ensures consistent state across all operations

Frontend: No special handling needed, backend manages singleton internally.
```

6.3 Factory Method Pattern

```
Backend Location: backend/src/main/java/com/smartedit/patterns/factory/
Files:
- DocumentFactory.java (abstract)
- TextDocumentFactory.java
- MarkdownDocumentFactory.java
- HTMLDocumentFactory.java

API: POST /api/file/new
Request: { type: 'txt' | 'md' | 'html' }
Response: { id, name, content, format }

Frontend: Call when user clicks 'New File', pass selected type.
```

6.4 Decorator Pattern

```
Backend Location: backend/src/main/java/com/smartedit/patterns/decorator/
Files:
- TextComponent.java (interface)
- PlainText.java
- BoldDecorator.java, ItalicDecorator.java, UnderlineDecorator.java

API: POST /api/editor/format
```

```
Request: { text, startPos, endPos, format: 'bold' | 'italic' | 'underline' }
Response: { content } (updated)

Frontend: User selects text, clicks format button, backend applies decorator.
```

7. Step-by-Step Implementation Checklist

Week 1: Project Setup

■ Create backend Spring Boot project
■ Create frontend React project
■ Setup CORS configuration
■ Test basic connectivity (hello world endpoint)
■ Create Document model class

Week 2: Command Pattern

■ Create Command interface
■ Implement InsertTextCommand
■ Implement DeleteTextCommand
■ Implement CommandManager with undo/redo stacks
■ Create EditorController with undo/redo endpoints
■ Create commandService.js in frontend
■ Add Undo/Redo buttons to toolbar
■ Test: Type text, undo, redo

Week 3: Memento Pattern

■ Create DocumentMemento class
■ Add createMemento() and restore() to Document
■ Implement MementoManager
■ Create SnapshotController
■ Create snapshotService.js in frontend
■ Add auto-save with useEffect timer
■ Create RestorePoints modal component
■ Test: Auto-save, view snapshots, restore

Week 4: Strategy Pattern

■ Create FileSaveStrategy interface
■ Implement TextSaveStrategy
■ Implement MarkdownSaveStrategy (with conversion logic)
■ Implement HTMLSaveStrategy (with HTML wrapper)
■ Implement FileManager
■ Create FileController with save endpoint
■ Create fileService.js in frontend
■ Create SaveAs modal with format dropdown
■ Test: Save files in different formats

Week 5: Observer Pattern

■ Create DocumentObserver interface

■ Implement StatusBarObserver
■ Implement AutoSaveObserver
■ Add observer list to Document class
■ Modify Document.setContent() to notify observers
■ Update API responses to include observer data
■ Update StatusBar component to show word/char count
■ Test: Type text, verify stats update

Week 6: Singleton + Factory + Decorator

■ Create EditorManager singleton
■ Create DocumentFactory classes
■ Create Decorator classes for text formatting
■ Wire all patterns together
■ Add remaining API endpoints
■ Complete frontend components
■ Integration testing

Week 7-8: Polish & Documentation

■ Bug fixes
■ Error handling
■ Unit tests for each pattern
■ Integration tests
■ UML diagrams
■ README documentation
■ Final presentation preparation

8. Important Notes & Best Practices

8.1 Backend Best Practices

• Keep all business logic in backend
• Use @Service layer for complex operations
• Use DTOs for API requests/responses
• Add proper exception handling (@ControllerAdvice)
• Use @CrossOrigin for CORS or configure globally

8.2 Frontend Best Practices

• Keep components small and focused
• Use separate service files for API calls
• Handle loading states and errors
• Use async/await for cleaner async code
• No business logic in frontend - only UI logic

8.3 Testing Strategy

• Unit test each pattern class independently
• Use JUnit for backend tests
• Test API endpoints with Postman or curl
• Manual UI testing for user workflows

8.4 Common Pitfalls to Avoid

✗ Implementing patterns in frontend (all patterns in backend!)

✗ Not using interfaces for patterns

✗ Forgetting CORS configuration

✗ Not testing undo/redo thoroughly

✗ Hardcoding file paths (use proper storage)

8.5 Running the Application

```
Backend (Port 8080):
cd backend
mvn spring-boot:run

Frontend (Port 3000):
cd frontend
npm start

Access: http://localhost:3000
```

Questions or Need Help?

Mir Bedirhan KAYGUSUZ - S028260
Özgür Tuna Yavuz - S024224

CS434 Software Design Patterns
Özye■in University