Лабораторная работа №22

Тема: Анализ архитектурных шаблонов MVC, MVP, MVVM

Цель работы: Закрепить теоретические знания по архитектурным шаблонам. Освоить методы сравнительного анализа вариантов архитектурных шаблонов.

Длительность: 4 часа

Теоретический материал

Архитектурные паттерны позволяют придерживаться различных правил «правильного» кода. Один из них – это разграничение логики у различных частей программы, что упрощает их дальнейшую разработку и поддержку.

В данной лабораторной работе будут детально рассмотрены паттерны MVC, MVP и MVVM.

Данный паттерны являются прямой эволюцией друг друга, а именно MVC → MVP → MVVM. В зависимости от требований проекта может реализоваться как классический MVC, так и более современный MVVM.

Паттерн MVC или Model-View-Controller характеризуется простой реализацией, четкой разделением ответственности, подходит для простых проектов. В данном паттерне вся логика приложения реализовано в Model.

Паттерн MVP или Model-View-Presenter характеризуется тем, что Presenter становиться посредником между Model и View, четкое разделение ответственности, подходит для сложных проектов с бизнес-логикой. В данном паттерне вся логика приложения реализовано в Presenter.

Паттерн MVVM или Model-View-ViewModel характеризуется тем, ViewModel хранит состояние UI, имеет механизм Data-Binding (автоматическая синхронизация между View и ViewModel). четкое разделение ответственности, подходит для проектов со сложным UI-интерфейсом.

Понимание и умение идентифицировать эти шаблоны в реальном коде — ключевой навык для анализа, рефакторинга и поддержки сложных программных систем.

Задание для лабораторной работы

Номер варианта для выполнения практической работы совпадает с порядковым номером учащегося в журнале для подгруппы.

Задание 1. Необходимо провести анализ архитектурного шаблона по варианту (табл. 1). После перечислить источники, использованные для анализа. Паттерн опишите по следующим критериям:

- Основная роль View
- Роль "посредника" (Controller/Presenter/ViewModel)
- Направление взаимодействия
- Подписка на изменения данных
- Наличие связи View -> Посредник
- Наличие связи Посредник -> View
- Тестируемость логики
- Сложность реализации
- Где чаще применяется

Таблица 1 – Список вариантов

Вариант	Паттерн
1	MVC
2	MVP
3	MVVM
4	MVC
5	MVP
6	MVVM
7	MVC
8	MVP
9	MVVM
10	MVC
11	MVP
12	MVVM
13	MVC
14	MVP

Задание 2. Необходимо по номеру варианта проанализировать ситуацию и определить:

- Какой архитектурный паттерн (MVC, MVP, MVVM) подошел бы лучшего всего? Обоснуйте свой ответ.

- Какой архитектурный паттерн (MVC, MVP, MVVM) был бы наименее удобен и почему?

Обоснуйте свой выбор, указав преимущества выбранного паттерна и недостатки других для данного конкретного случая.

Вариант 1

Представьте, что вы разрабатываете окно настроек приложения. Пользователь может изменить язык интерфейса, цветовую тему и уровень звука. После каждого изменения параметра его выбор должен немедленно сохраняться в базу данных, а интерфейс — частично перерисовываться (например, меняться цвет кнопок при смене темы).

Вариант 2

Приложение позволяет инспекторам заполнять формы проверок в полевых условиях без интернета. Данные сохраняются локально в SQLite, при появлении сети синхронизируются с сервером. UI сложный, с множеством различной валидаций.

Вариант 3

Необходимо быстро создать прототип приложения для демонстрации инвесторам. Функционал будет меняться ежедневно, важна скорость разработки.

Вариант 4

Одно приложение должно работать на iOS, Android и Web с максимальным переиспользованием кода. UI нативных платформ значительно отличается.

Вариант 5

Необходимо постоянно экспериментировать с UI: менять кнопки, формы, потоки данных. Изменения должны отображаться без перекомпиляции проекта.

Вариант 6

Приложение включает различные типы упражнений: выбор правильного варианта, заполнение пропусков, аудирование и разговорная практика. Упражнения должны проверяться автоматически, прогресс пользователя сохраняться и отображаться в реальном времени. Важна быстрая реакция интерфейса на действия пользователя.

Вариант 7

Приложение для складских работников, которые сканируют штрих-коды товаров, обновляют остатки, проводят инвентаризацию. Работает в условиях нестабильного интернета, требует оффлайн-работы с последующей синхронизацией.

Вариант 8

Есть приложение для заказа еды и доставки. В нем нужно реализовать: выбор ресторанов, меню, корзина, оплата, трекинг доставки, push-уведомления, рейтинги и отзывы.

Вариант 9

Необходимо разработать простое приложение типа "todo list" или "калькулятор" для изучения основ программирования.

Вариант 10

Система приложения для управлений проектами типа Jira позволяет: создавать задачи, назначать исполнителей, фиксировать прогресс, создавать диаграммы Ганта и различные отчеты. Есть возможность интеграции с системой контроля версиями (Git).

Вариант 11

Приложение для медитации представляет собой трекинг сессий, которое предоставляет пользователю возможность послушать аудиогиды медитации, персональные рекомендации на основе прослушанных медитаций, интеграция

с умными часами и другими гаджетами, отслеживающими здоровья пользователя. Приложение имеет красивый и спокойный UI с анимациями.

Вариант 12

Приложение для планирования путешествий позволяет искать и бронировать отели, авиабилеты. Также имеет картотеку различных туристических маршрутов со статическими картами маршрутов (картинки). Пользователь может получать уведомления на почту по расписанию.

Вариант 13

Необходимо разработать простую систему аутентификации, в которой будет модуль входа/регистрации с базовой валидацией без сложных сценариев.

Вариант 14

Необходимо разработать простой консольной интерфейс для утилиты обработки файлов.

Задание 3. По номеру варианта нужно проанализировать код и определить:

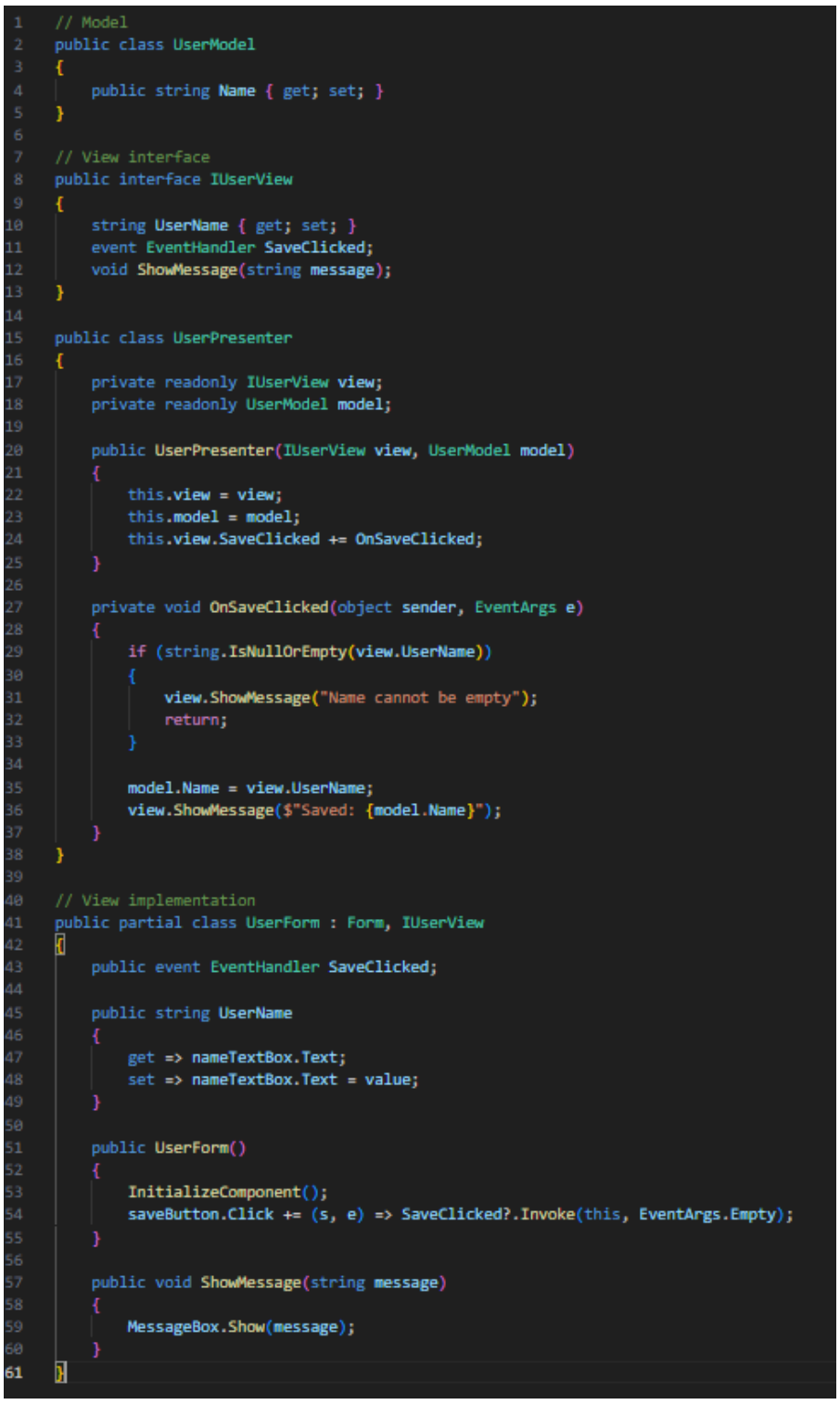
- Какой архитектурный паттерн (или их смесь) использован?
- Обосновать свой вывод, указав на характерные признаки паттерна в коде

Все варианты описаны в Приложении А.

ПРИЛОЖЕНИЕ А
(варианты для 3 задания)

```java
// CounterModel.java
public class CounterModel {
    private int count = 0;
    public void increment() { count++; }
    public int getCount() { return count; }
}

// CounterView.java (Swing)
public class CounterView extends JFrame {
    private JLabel countLabel;
    private JButton button;
    // View хранит ссылку на Controller
    private CounterController controller;

    public CounterView() {
        // ... инициализация UI элементов ...
        button.addActionListener(e -> controller.onButtonClicked());
    }
    public void setCount(String text) {
        countLabel.setText(text);
    }
    public void setController(CounterController controller) {
        this.controller = controller;
    }
}

// CounterController.java
public class CounterController {
    private CounterModel model;
    private CounterView view;

    public CounterController(CounterModel model, CounterView view) {
        this.model = model;
        this.view = view;
        this.view.setController(this); // View знает о Controller
        updateView(); // Первоначальное обновление
    }
    // Обработчик события от View
    public void onButtonClicked() {
        model.increment();
        updateView();
    }
    // Controller явно обновляет View
    private void updateView() {
        view.setCount("Count: " + model.getCount());
    }
}
```

Рисунок А.1 - Вариант 1



```
// Model
public class UserModel
{
    public string Name { get; set; }
}

// View interface
public interface IUserView
{
    string UserName { get; set; }
    event EventHandler SaveClicked;
    void ShowMessage(string message);
}

public class UserPresenter
{
    private readonly IUserView view;
    private readonly UserModel model;

    public UserPresenter(IUserView view, UserModel model)
    {
        this.view = view;
        this.model = model;
        this.view.SaveClicked += OnSaveClicked;
    }

    private void OnSaveClicked(object sender, EventArgs e)
    {
        if (string.IsNullOrEmpty(view.UserName))
        {
            view.ShowMessage("Name cannot be empty");
            return;
        }

        model.Name = view.UserName;
        view.ShowMessage($"Saved: {model.Name}");
    }
}

// View implementation
public partial class UserForm : Form, IUserView
{
    public event EventHandler SaveClicked;

    public string UserName
    {
        get => nameTextBox.Text;
        set => nameTextBox.Text = value;
    }

    public UserForm()
    {
        InitializeComponent();
        saveButton.Click += (s, e) => SaveClicked?.Invoke(this, EventArgs.Empty);
    }

    public void ShowMessage(string message)
    {
        MessageBox.Show(message);
    }
}
```

Рисунок А.2 - Вариант 2

```kotlin
// Contract определяет интерфейсы View и Presenter
interface MainContract {
    interface View {
        fun showMessage(message: String)
    }
    interface Presenter {
        fun onButtonClicked()
    }
}

class MainPresenter(private val view: MainContract.View) : MainContract.Presenter {
    override fun onButtonClicked() {
        // Presenter решает, что показать
        view.showMessage("Button clicked!")
    }
}

class MainActivity : AppCompatActivity(), MainContract.View {
    lateinit var presenter: MainPresenter

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        presenter = MainPresenter(this) // View создает Presenter

        findViewById<Button>(R.id.button).setOnClickListener {
            presenter.onButtonClicked() // View делегирует событие
        }
    }
    override fun showMessage(message: String) {
        findViewById<TextView>(R.id.textView).text = message
    }
}
```

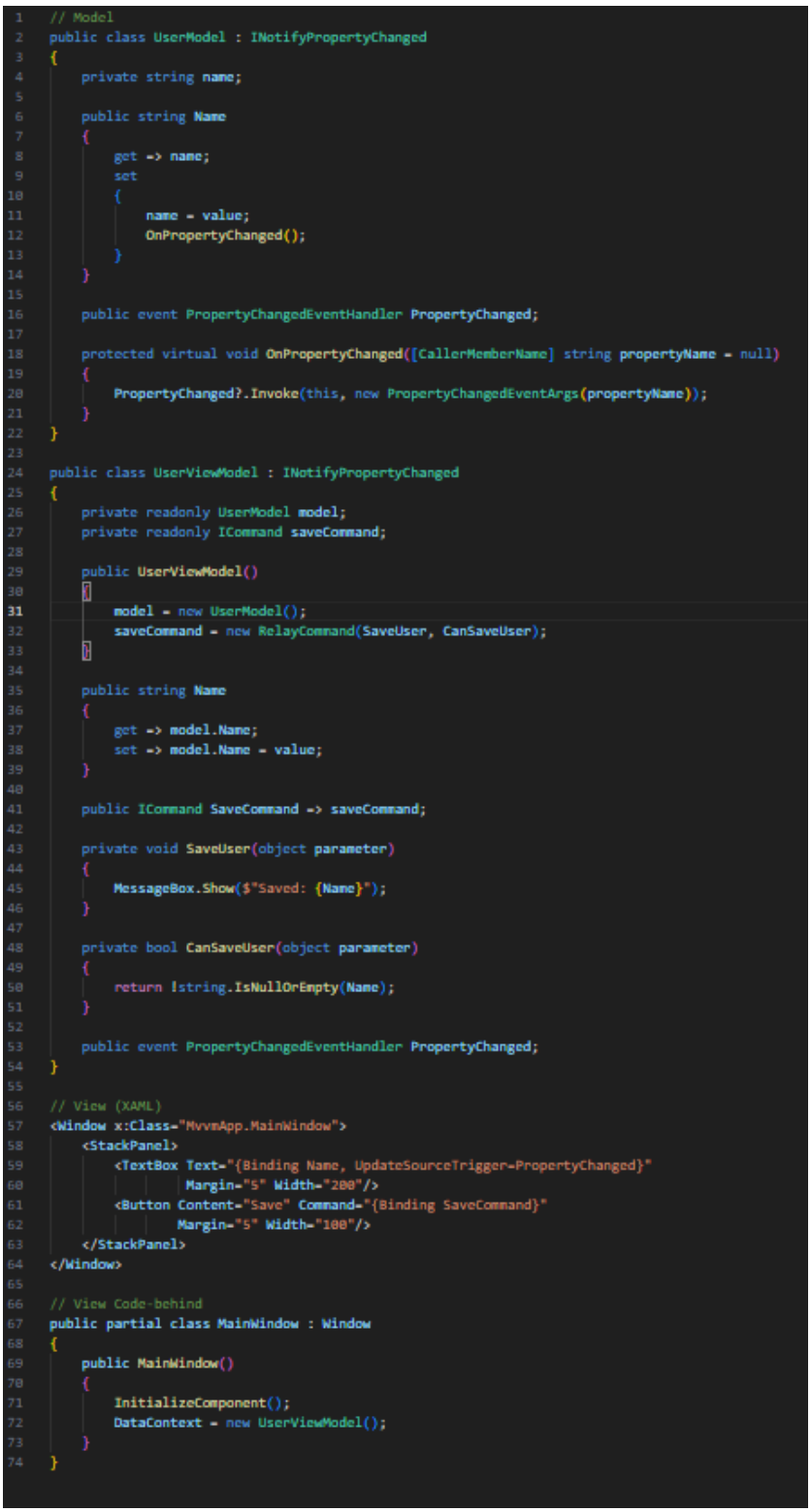
Рисунок А.3 – Вариант 3

```java
public class MainViewModel {
    private final StringProperty message = new SimpleStringProperty();

    public StringProperty messageProperty() {
        return message;
    }
    public void onButtonClicked() {
        message.set("Button clicked!");
    }
}

public class MainView {
    @FXML private TextField textField;
    @FXML private Button button;

    private MainViewModel viewModel = new MainViewModel();

    public void initialize() {
        // Привязка свойства текстового поля к свойству ViewModel
        textField.textProperty().bind(viewModel.messageProperty());
        // Привязка команды кнопки к методу ViewModel
        button.setOnAction(event -> viewModel.onButtonClicked());
    }
}
```

Рисунок А.4 – Вариант 4

```java
// Model
public class UserModel {
    private String name;
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
}

// View
public class UserView extends JFrame {
    private JTextField nameField = new JTextField(20);
    private JButton saveButton = new JButton("Save");

    public UserView() {
        setLayout(new FlowLayout());
        add(new JLabel("Name:"));
        add(nameField);
        add(saveButton);
    }

    public void setController(UserController controller) {
        saveButton.addActionListener(e -> controller.onSave());
    }

    public String getName() { return nameField.getText(); }
    public void setName(String name) { nameField.setText(name); }
}


public class UserController {
    private UserModel model;
    private UserView view;

    public UserController(UserModel model, UserView view) {
        this.model = model;
        this.view = view;
        view.setController(this);
    }

    public void onSave() {
        model.setName(view.getName());
        JOptionPane.showMessageDialog(null, "Saved: " + model.getName());
    }
}
```

Рисунок А.5 – Вариант 5



```
// Model
public class UserModel : INotifyPropertyChanged
{
    private string name;

    public string Name
    {
        get => name;
        set
        {
            name = value;
            OnPropertyChanged();
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged([CallerMemberName] string propertyName = null)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

public class UserViewModel : INotifyPropertyChanged
{
    private readonly UserModel model;
    private readonly ICommand saveCommand;

    public UserViewModel()
    {
        model = new UserModel();
        saveCommand = new RelayCommand(SaveUser, CanSaveUser);
    }

    public string Name
    {
        get => model.Name;
        set => model.Name = value;
    }

    public ICommand SaveCommand => saveCommand;

    private void SaveUser(object parameter)
    {
        MessageBox.Show($"Saved: {Name}");
    }

    private bool CanSaveUser(object parameter)
    {
        return !string.IsNullOrEmpty(Name);
    }

    public event PropertyChangedEventHandler PropertyChanged;
}

// View (XAML)
<Window x:Class="MvvmApp.MainWindow">
    <StackPanel>
        <TextBox Text="{Binding Name, UpdateSourceTrigger=PropertyChanged}"
                 Margin="5" Width="200"/>
        <Button Content="Save" Command="{Binding SaveCommand}"
                Margin="5" Width="100"/>
    </StackPanel>
</Window>

// View Code-behind
public partial class MainWindow : Window
{
    public MainWindow()
    {
        InitializeComponent();
        DataContext = new UserViewModel();
    }
}
```

Рисунок А.6 – Вариант 6

```
// Model (could be a class or just data structure)
class User {
    constructor(name = '') {
        this.name = name;
    }
}

const UserComponent = {
    template: `
        <div>
            <input v-model="user.name" placeholder="Enter name">
            <button @click="saveUser">Save</button>
            <p>{{ message }}</p>
        </div>
    `,
    data() {
        return {
            user: new User(),
            message: ''
        };
    },
    methods: {
        saveUser() {
            if (!this.user.name.trim()) {
                this.message = 'Name cannot be empty';
                return;
            }

            this.message = `Saved: ${this.user.name}`;
            // Typically would call API here
        }
    }
};

// Main app
new Vue({
    el: '#app',
    components: { UserComponent },
    template: '<UserComponent/>'
});
```

Рисунок А.7 – Вариант 7

```php
<?php
// Model (UserModel.php)
class UserModel {
    private $name;

    public function setName($name) {
        $this->name = $name;
    }

    public function getName() {
        return $this->name;
    }
}

// Controller (UserController.php)
class UserController {
    private $model;

    public function __construct($model) {
        $this->model = $model;
    }

    public function handleRequest() {
        if ($_SERVER['REQUEST_METHOD'] === 'POST') {
            $this->model->setName($_POST['name']);
        }
    }

    public function getUserData() {
        return ['name' => $this->model->getName()];
    }
}

// View (user_view.php)
$controller = new UserController($model);
$controller->handleRequest();
$data = $controller->getUserData();
?>
<form method="POST">
    <input type="text" name="name" value="<?= htmlspecialchars($data['name']) ?>">
    <button type="submit">Save</button>
</form>
```

Рисунок А.8 – Вариант 8

```
// Model
export class UserModel {
  private name: string = '';

  setName(name: string): void {
    this.name = name;
  }

  getName(): string {
    return this.name;
  }
}

// View interface
export interface UserView {
  displayUserName(name: string): void;
  showError(message: string): void;
}

// Presenter
export class UserPresenter {
  constructor(
    private view: UserView,
    private model: UserModel
  ) {}

  saveUser(name: string): void {
    if (!name.trim()) {
      this.view.showError('Name cannot be empty');
      return;
    }

    this.model.setName(name);
    this.view.displayUserName(this.model.getName());
  }
}

// Component implementing View
@Component({
  selector: 'app-user',
  template: `
    <input [(ngModel)]="userName" placeholder="Enter name">
    <button (click)="onSave()">Save</button>
    <p>{{ displayText }}</p>
  `
})
export class UserComponent implements UserView, OnInit {
  userName: string = '';
  displayText: string = '';

  private presenter: UserPresenter;

  constructor() {
    const model = new UserModel();
    this.presenter = new UserPresenter(this, model);
  }

  onSave(): void {
    this.presenter.saveUser(this.userName);
  }

  displayUserName(name: string): void {
    this.displayText = `Saved: ${name}`;
  }

  showError(message: string): void {
    alert(message);
  }
}
```

Рисунок А.9 – Вариант 9

```
// Model
class UserModel {
    constructor() {
        this.name = '';
    }

    setName(name) {
        this.name = name;
    }

    getName() {
        return this.name;
    }
}

// View
class UserView {
    constructor() {
        this.nameInput = document.getElementById('name-input');
        this.saveButton = document.getElementById('save-btn');
        this.display = document.getElementById('display');
    }

    bindSave(handler) {
        this.saveButton.addEventListener('click', handler);
    }

    getName() {
        return this.nameInput.value;
    }

    setName(name) {
        this.nameInput.value = name;
    }

    updateDisplay(name) {
        this.display.textContent = `Saved: ${name}`;
    }
}

// Controller
class UserController {
    constructor(model, view) {
        this.model = model;
        this.view = view;

        this.view.bindSave(this.handleSave.bind(this));
    }

    handleSave() {
        this.model.setName(this.view.getName());
        this.view.updateDisplay(this.model.getName());
    }
}

```

Рисунок А.10 – Вариант 10

```java
// Contract interface
public interface UserContract {
    interface View {
        void showUserName(String name);
        void showError(String message);
    }

    interface Presenter {
        void saveUser(String name);
        void loadUser();
    }
}

// Presenter implementation
public class UserPresenter implements UserContract.Presenter {
    private UserContract.View view;
    private UserModel model;

    public UserPresenter(UserContract.View view, UserModel model) {
        this.view = view;
        this.model = model;
    }

    @Override
    public void saveUser(String name) {
        if (name.isEmpty()) {
            view.showError("Name cannot be empty");
            return;
        }
        model.setName(name);
        view.showUserName(name);
    }

    @Override
    public void loadUser() {
        view.showUserName(model.getName());
    }
}

// View implementation (Activity)
public class UserActivity extends AppCompatActivity implements UserContract.View {
    private UserPresenter presenter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_user);

        UserModel model = new UserModel();
        presenter = new UserPresenter(this, model);

        Button saveButton = findViewById(R.id.save_button);
        saveButton.setOnClickListener(v -> {
            EditText nameEditText = findViewById(R.id.name_edit_text);
            presenter.saveUser(nameEditText.getText().toString());
        });
    }

    @Override
    public void showUserName(String name) {
        TextView nameTextView = findViewById(R.id.name_text_view);
        nameTextView.setText(name);
    }

    @Override
    public void showError(String message) {
        Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
    }
}
```

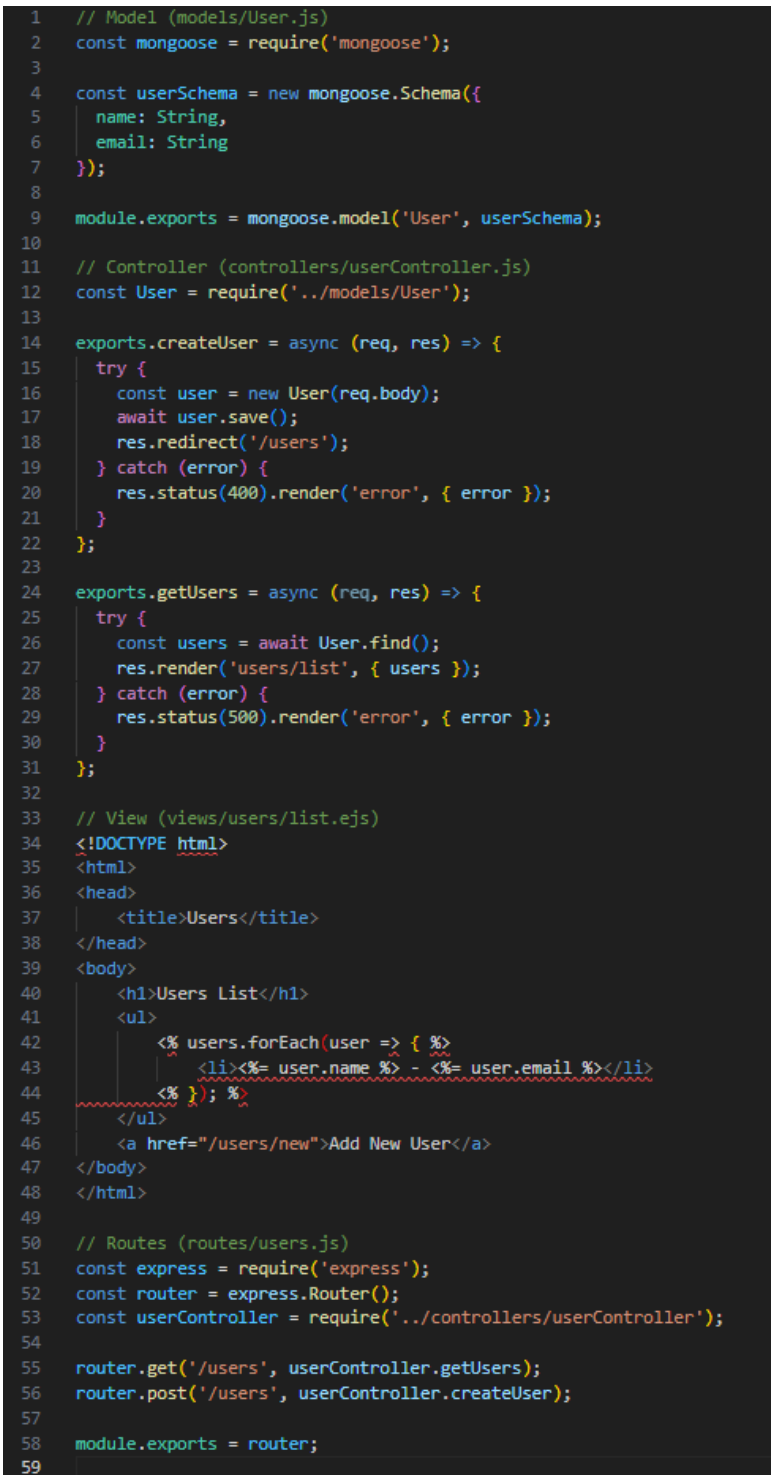
Рисунок А.11 – Вариант 11

```
// ViewModel (Custom Hook)
const useUserViewModel = () => {
  const [users, setUsers] = useState([]);
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');

  const loadUsers = useCallback(async () => {
    setLoading(true);
    setError('');
    try {
      // Simulate API call
      await new Promise(resolve => setTimeout(resolve, 1000));
      setUsers([
        { id: 1, name: 'John Doe', email: 'john@example.com' },
        { id: 2, name: 'Jane Smith', email: 'jane@example.com' },
      ]);
    } catch (err) {
      setError('Failed to load users');
    } finally {
      setLoading(false);
    }
  }, []);

  const addUser = useCallback((user) => {
    setUsers(prev => [...prev, { ...user, id: Date.now() }]);
  }, []);

  return {
    users,
    loading,
    error,
    loadUsers,
    addUser
  };
};

// View
const UserListScreen = () => {
  const { users, loading, error, loadUsers } = useUserViewModel();
  const [isAdding, setIsAdding] = useState(false);

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

  if (loading) {
    return (
      <View style={styles.center}>
        <ActivityIndicator size="large" />
      </View>
    );
  }

  if (error) {
    return (
      <View style={styles.center}>
        <Text style={styles.error}>{error}</Text>
        <Button title="Retry" onPress={loadUsers} />
      </View>
    );
  }

  return (
    <View style={styles.container}>
      <FlatList
        data={users}
        keyExtractor={item => item.id.toString()}
        renderItem={({ item }) => (
          <View style={styles.item}>
            <Text style={styles.name}>{item.name}</Text>
            <Text style={styles.email}>{item.email}</Text>
          </View>
        )}
      />
      <Button title="Add User" onPress={() => setIsAdding(true)} />

      <Modal visible={isAdding} animationType="slide">
        <AddUserView
          onAdd={addUser}
          onCancel={() => setIsAdding(false)}
        />
      </Modal>
    </View>
  );
};
```

Рисунок А.12 – Вариант 12 (1 часть)

```
86
87  // Add User Component
88  const AddUserView = ({ onAdd, onCancel }) => {
89    const [name, setName] = useState('');
90    const [email, setEmail] = useState('');
91
92    const handleAdd = () => {
93      if (name && email) {
94        onAdd({ name, email });
95        onCancel();
96      }
97    };
98
99    return (
100     <View style={styles.modalContainer}>
101       <TextInput
102         style={styles.input}
103         placeholder="Name"
104         value={name}
105         onChangeText={setName}
106       />
107       <TextInput
108         style={styles.input}
109         placeholder="Email"
110         value={email}
111         onChangeText={setEmail}
112         keyboardType="email-address"
113       />
114       <View style={styles.buttonRow}>
115         <Button title="Cancel" onPress={onCancel} />
116         <Button title="Add" onPress={handleAdd} />
117       </View>
118     </View>
119   );
120 };
```

Рисунок А.13 – Вариант 12 (2 часть)



```
// Model (models/User.js)
const mongoose = require('mongoose');

const userSchema = new mongoose.Schema({
  name: String,
  email: String
});

module.exports = mongoose.model('User', userSchema);

// Controller (controllers/userController.js)
const User = require('../models/User');

exports.createUser = async (req, res) => {
  try {
    const user = new User(req.body);
    await user.save();
    res.redirect('/users');
  } catch (error) {
    res.status(400).render('error', { error });
  }
};

exports.getUsers = async (req, res) => {
  try {
    const users = await User.find();
    res.render('users/list', { users });
  } catch (error) {
    res.status(500).render('error', { error });
  }
};

// View (views/users/list.ejs)
<!DOCTYPE html>
<html>
<head>
    <title>Users</title>
</head>
<body>
    <h1>Users List</h1>
    <ul>
        <% users.forEach(user => { %>
            <li><%= user.name %> - <%= user.email %></li>
        <% }); %>
    </ul>
    <a href="/users/new">Add New User</a>
</body>
</html>

// Routes (routes/users.js)
const express = require('express');
const router = express.Router();
const userController = require('../controllers/userController');

router.get('/users', userController.getUsers);
router.post('/users', userController.createUser);

module.exports = router;
```

Рисунок А.14 – Вариант 13

```java
// Model
public class CalculatorModel {
    private double result;

    public void add(double a, double b) {
        result = a + b;
    }

    public void subtract(double a, double b) {
        result = a - b;
    }

    public double getResult() {
        return result;
    }
}

// View Interface
public interface CalculatorView {
    void setResult(double result);
    void showError(String message);
    double getFirstNumber();
    double getSecondNumber();
}

// Presenter
public class CalculatorPresenter {
    private final CalculatorView view;
    private final CalculatorModel model;

    public CalculatorPresenter(CalculatorView view) {
        this.view = view;
        this.model = new CalculatorModel();
    }

    public void onAdd() {
        try {
            double a = view.getFirstNumber();
            double b = view.getSecondNumber();
            model.add(a, b);
            view.setResult(model.getResult());
        } catch (NumberFormatException e) {
            view.showError("Invalid input");
        }
    }

    public void onSubtract() {
        try {
            double a = view.getFirstNumber();
            double b = view.getSecondNumber();
            model.subtract(a, b);
            view.setResult(model.getResult());
        } catch (NumberFormatException e) {
            view.showError("Invalid input");
        }
    }
}
```

Рисунок А.15 – Вариант 14 (1 часть)

```
}

// View Implementation
public class CalculatorApp extends Application implements Calculato
    private TextField firstNumberField;
    private TextField secondNumberField;
    private Label resultLabel;
    private CalculatorPresenter presenter;

    @Override
    public void start(Stage stage) {
        presenter = new CalculatorPresenter(this);

        VBox root = new VBox(10);
        firstNumberField = new TextField();
        secondNumberField = new TextField();
        Button addButton = new Button("Add");
        Button subtractButton = new Button("Subtract");
        resultLabel = new Label("Result: ");

        addButton.setOnAction(e -> presenter.onAdd());
        subtractButton.setOnAction(e -> presenter.onSubtract());

        root.getChildren().addAll(
            new Label("First Number:"), firstNumberField,
            new Label("Second Number:"), secondNumberField,
            addButton, subtractButton, resultLabel
        );

        stage.setScene(new Scene(root, 300, 250));
        stage.show();
    }

    @Override
    public double getFirstNumber() {
        return Double.parseDouble(firstNumberField.getText());
    }

    @Override
    public double getSecondNumber() {
        return Double.parseDouble(secondNumberField.getText());
    }

    @Override
    public void setResult(double result) {
        resultLabel.setText("Result: " + result);
    }

    @Override
    public void showError(String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setContentText(message);
        alert.show();
    }
}
```

Рисунок А.16 – Вариант 14 (2 часть)