|  |  |
| --- | --- |
| C:\Users\7636~1\AppData\Local\Temp\Rar$DRa0.802\герб_синий.jpg | |
| МИНОБРНАУКИ РОССИИ | |
| Федеральное государственное бюджетное образовательное учреждениевысшего образования"МИРЭА - Российский технологический университет"РТУ МИРЭА | |
| Институт информационных технологий | |
| Кафедра *МОСИТ* | |

|  |  |
| --- | --- |
| **РЕФЕРАТ** | |
| **по дисциплине Модели и средства разработки программных интерфейсов** | |
| «Фронтальная подсистема автоматизированной системы электронного документооборота» | |
|  | |
| Выполнил студент группы *ИВБО-02-16* | *Кубанцев В.А.* |
| Принял преподаватель | *Петренко А.А.* |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_\_»\_\_\_\_\_\_\_2020 г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_\_»\_\_\_\_\_\_\_2020 г. | *(подпись руководителя)* |

Москва 2020

Оглавление

[Введение 3](#_Toc34216047)

[1. Цель 4](#_Toc34216048)

[2. Ход выполнения работы 4](#_Toc34216049)

[3. Выводы 6](#_Toc34216050)

[4. Список использованных информационных источников 7](#_Toc34216051)

Введение

Web-интерфейс — [веб-страница](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D1%82%D1%80%D0%B0%D0%BD%D0%B8%D1%86%D0%B0) или совокупность веб-страниц, предоставляющая [пользовательский интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8F) для взаимодействия с сервисом или устройством посредством протокола [HTTP](https://ru.wikipedia.org/wiki/HTTP) и [веб-браузера](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80).

Web-интерфейсы получили широкое распространение в связи с ростом популярности [всемирной паутины](https://ru.wikipedia.org/wiki/WWW) и соответственно — повсеместного распространения [веб-браузеров](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80).

Одним из основных требований к веб-интерфейсам является их одинаковый внешний вид и одинаковая функциональность при работе в различных браузерах.

1. Цель

Ознакомиться и использовать, и привести примеры использования Web-интерфейсов из ВКР

1. Ход выполнения работы

Spring — легкий, но мощный фреймворк для разработки приложений. В более широком смысле, Spring — хорошо проработанный инструмент, который поддерживает несколько веб-приложений, использующих Java в качестве языка программирования.

До появления этого фреймворка, приложения разрабатывались с помощью стандартов JEE. С ними можно развертывать приложения на любом сервере приложений JEE.

Но было и несколько проблем, включая:

* Код становился все сложнее по мере развития приложения;
* Производительность системы страдала от тяжести приложений;
* Проблема поиска компонента.

**Шаблон проектирования Backend for Frontend (BFF)**

Первоначально приложение может быть нацелено на настольный веб-интерфейс. Как правило, параллельно разрабатывается серверная служба, которая предоставляет функции, необходимые для этого пользовательского интерфейса. По мере роста пользовательской базы приложения разрабатывается мобильное приложение, которое должно взаимодействовать с тем же бэкэндом. Бэкэнд-сервис становится бэкэндом общего назначения, обслуживающим требования как настольных, так и мобильных интерфейсов.  
  
Но возможности мобильного устройства существенно отличаются от настольного браузера с точки зрения размера экрана, производительности и ограничений отображения. В результате требования к бэкенду мобильного приложения отличаются от требований к веб-интерфейсу рабочего стола.

Эти различия приводят к конкурирующим требованиям к бэкэнду. Серверная часть требует регулярных и значительных изменений для обслуживания как настольного веб-интерфейса, так и мобильного приложения. Часто отдельные интерфейсные команды работают над каждым фронтендом, в результате чего бэкэнд становится узким местом в процессе разработки. Конфликтующие требования к обновлению и необходимость поддерживать работу службы для обоих интерфейсов могут привести к тому, что вы потратите много усилий на один развертываемый ресурс.

Поскольку деятельность по разработке сосредоточена на серверной службе, может быть создана отдельная команда для управления и обслуживания серверной части. В конечном счете, это приводит к разрыву между интерфейсной и серверной командами разработки, возлагая бремя на серверную команду, чтобы сбалансировать конкурирующие требования различных команд пользовательского интерфейса. Если одной интерфейсной команде требуются изменения в бэкэнде, эти изменения должны быть проверены другими интерфейсными командами, прежде чем они могут быть интегрированы в бэкэнд.

Решением проблем – создание одного сервера на пользовательский интерфейс. Тонкая настройка поведения и производительности каждого бэкенда, чтобы наилучшим образом соответствовать потребностям среды frontend, не беспокоясь о влиянии на другие интерфейсные интерфейсы.

**Плюсы шаблона BFF**

1. **Фильтрация**. Клиент получает только то, что ему нужно, ни больше ни меньше.
2. **Агрегация**. Не нужно тратить клиентскую сеть и батарею, чтобы делать множественные аякс-запросы. Заметный выигрыш по скорости за счет того, что открытие соединения — это дорогая операция.
3. **Кэширование**. Ваш повторный агрегированный вызов не будет лишний раз никого дергать, а просто вернет 304 Not Modified.
4. **Сокрытие** данных. Например, у вас могут быть токены, которые нужны между бэкендами и не должны попадать на клиент. У клиента может не быть прав даже знать о существовании этих токенов, не говоря уже об их содержимом.
5. **Микросервисы**. Если у вас на бэке монолит, то BFF — это первый шаг к микросервисам.

**Минусы шаблона BFF**

1. Повышение **сложности**. Любая абстракция — это еще один слой, который необходимо кодить, деплоить, поддерживать. Еще одна движущаяся часть механизма, которая может сбоить.
2. **Дублирование** индексов. Например, несколько ендпойнтов могут выполнять один и тот же тип агрегаций.
3. BFF — это **пограничный слой**, который должен поддерживать общую маршрутизацию, ограничения прав пользователей, ведение журнала запросов и т. д.

**Особенности Spring**

1. Легкость

Фреймворк Spring очень легкий с точки зрения размера и функциональности. Все благодаря его реализации POJO (*Plain Old Java Object*), которая не требует наследования каких-либо классов или реализации каких-либо интерфейсов.

1. Аспектно-Ориентированное Программирование (АОП)

Важная часть Spring. Аспектно-Ориентированное Программирование используется для отделения сквозной функциональности (например, логирования, безопасности, и тд) от бизнес-логики.

1. Управление Транзакциями

Используется для объединения нескольких API, управляющими транзакциями, и координации транзакций объектов Java. Кроме того, оно не связано со средой J2EE и используется с бесконтейнерными средами.

1. Контейнер

Spring проектирует и управляет жизненным циклом и настройками объектов приложения.

1. Внедрение Зависимостей

Эта фича позволяет разрабатывать слабо связанные приложения. Таким образом, стало гораздо проще проводить их модульное тестирование. Это также позволяет разработчику по необходимости менять модули.

1. Интеграция с Другими Фреймворками

Spring хорош еще и тем, что не пытается решать проблемы, которые уже решены. Он просто обеспечивает интеграцию фреймворков, решающих более сложные проблемы. К примеру, IBATIS, Hibernate, Toplink и тд.

**Реализация контроллеров**

Используя аннотацию @Controller Spring’a производим настройку обработки водящих от пользователей запросов:

* @Autowired указывается bean.
* *@*[Controller](https://habrahabr.ru/users/controller/) определяет класс как контроллер Spring MVC.
* *@*RequestMapping указывает, что все методы в данном контроллере относятся к URL-адресу "/".
* *@*GetMapping используется для объявления метода *getIndex()* как метода для обработки GET-запросов по HTTP.
* @PostMappingиспользуется для обработки POST-запросов по HTTP.

Контроллер MainController обрабатывает *Get* запросы по индексам */*, */index* и */files*.

@Controller

public class MainController {

@Autowired

private FileRepository fileRepository;

@GetMapping(value = { "/", "/index" })

public String index(Model model) {

return "index";

}

@GetMapping(value = "/listFile")

public String personList(Map<String, Object> model) {

Iterable<File> files = fileRepository.findAll();

model.put("files", files);

return "listFile";

}

}

public interface FileRepository extends JpaRepository<File, Long> {

Optional<File> findById(Long id);

Optional<File> findByAuthor(String username);

}

Контроллер UploadFileController позволяет пользователям получать список файлов, добавлять новые файлы и загружать собственные.

@Controller

public class UploadFileController {

@Autowired

private FileRepository fileRepository;

@Value ("${upload.path}")

private String uploadPath;

@GetMapping (value = "/addFile")

public String showAddPersonPage(Model model) {

FileForm fileForm = new FileForm();

model.addAttribute("fileForm", fileForm);

return "addFile";

}

@PostMapping (value = { "/addFile" })

public String saveFile(@RequestParam("filename") MultipartFile file,

@RequestParam String type,

@RequestParam String parent,

@RequestParam String author,

@RequestParam String editor) throws IOException {

File newFile = null;

if (file != null) {

java.io.File uploadDir = new java.io.File(uploadPath);

if (!uploadDir.exists()) {

uploadDir.mkdir();

}

String resultName = file.getOriginalFilename();

String size = Calc.getFileSize(file.getSize());

String path = uploadPath + resultName;

String date = String.valueOf(new Date().getTime());

newFile = new File(resultName, type, size, date, parent, author, editor, path);

file.transferTo(new java.io.File(uploadPath + resultName));

}

if (newFile != null)

fileRepository.save(newFile);

return "redirect:/listFile";

}

}

**Реализация отображения**

Для того, чтобы контроллеры смогли обеспечить необходимый функционал, помимо логики, также необходимо описать и визуальное представление страниц. Визуальное оформление выполняется с помощью языка разметки html и применение шаблонизатора thymeleaf.

Пример описания страницы Add File:

<!DOCTYPE HTML>

<html xmlns:th="http://www.thymeleaf.org">

<head>

<meta charset="UTF-8" />

<title>Add File</title>

<link rel="stylesheet" type="text/css" th:href="@{/css/style.css}"/>

</head>

<body>

<h1>Create a File:</h1>

<div>

<form th:action="@{/addFile}"

th:object="${fileForm}" method="POST"

enctype="multipart/form-data">

File:

<input type="file" th:field="\*{filename}" th:name="file"><br/>

Type:

<input type="text" th:field="\*{type}" placeholder="File/Folder"/><br/>

Parent:

<input type="text" th:field="\*{parent}" placeholder="folder X"/><br/>

Author:

<input type="text" th:field="\*{author}" placeholder="You username"/><br/>

Editor:

<input type="text" th:field="\*{editor}" placeholder="Бурыкин Иван"/><br/>

<input type="submit" value="Create" />

</form>

</div><br/>

</body>

</html>

1. Выводы

В результате выполнения лабораторной работы мной были получены навыки:

* Создания контроллеров на языке Java используя фреймворк Spring.
* Создания сущностей на языке Java используя фреймворк Spring.
* Создание и настройка подключения к Базе Данных на языке Java используя фреймворк Spring.

1. Список использованных информационных источников
2. Интернет – ресурс: <https://javarush.ru/groups/posts/2125-veb-prilozhenie-na-java> (Web application Java).
3. Интернет – ресурс: https://ru.wikipedia.org/wiki/Веб-интерфейс (Web-интерфейс).
4. Интернет – ресурс: <https://sohabr.net/habr/post/439044/> (описание преимуществ Spring).
5. Интернет – ресурс: https://habr.com/ru/post/455794/ (введение в Spring).