ГУАП

КАФЕДРА № 43

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | А. В. Фомин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1 |
| ПРИМЕНЕНИЕ ПРОГРАММНЫХ КАРКАСОВ ДЛЯ РАЗРАБОТКИ ПРИЛОЖЕНИЙ |
| по курсу: СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4232М |  |  |  | В. Ф. Губайдулин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

**Цель** **работы:**

Изучение программных каркасов (фреймворков) для разработки web приложений.

**Вариант:**

Разрабатываемое приложение ­– автоматизированная система управления составом футбольной команды.

Система будет позволять работать со списком команд, с игроками, играющими за определённую команду, а также с формациями – схемами игры.

**Ход работы:**

1) Для реализации back-end части разрабатываемого приложения был выбран программный каркас Spring Framework, реализуемый на языке программирования Java. Для реализации front-end части приложения был выбран каркас React, реализуемый на языке программирования JavaScript. В качестве системы управления базами данных был выбран MySQL.

2) Spring Framework – является общим названием для ряда небольших фреймворков, каждый из которых выполняет свою работу.

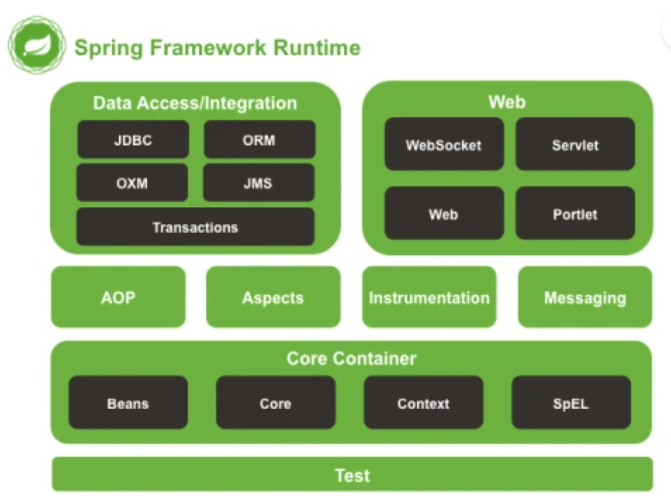


Рисунок 1 – Структура Java Spring Framework

Модульная структура позволяет подключать только необходимые модули. Spring может использоваться не только для разработки Web-приложений, но и для разработки консольных приложений. Центральной частью Spring является контейнер Inversion of Control, который предоставляет средства конфигурирования и управления объектами Java с помощью рефлексии. Контейнер отвечает за управление жизненным циклом объекта: создание объектов, вызов методов инициализации и конфигурирование объектов путём связывания их между собой. Объекты, создаваемые контейнером, также называются управляемыми объектами (beans).

Spring имеет собственную MVC-платформу веб-приложений. Spring MVC является фреймворком, ориентированным на запросы. В нем определены стратегические интерфейсы для всех функций современной запросно-ориентированной системы. Цель каждого интерфейса — быть простым и ясным.

React — JavaScript-библиотека с открытым исходным кодом для разработки пользовательских интерфейсов. React может использоваться для разработки одностраничных и мобильных приложений. Его цель — предоставить высокую скорость разработки, простоту и масштабируемость. React использует виртуальный DOM (англ. virtual DOM). React создаёт кэш-структуру в памяти, что позволяет вычислять разницу между предыдущим и текущим состояниями интерфейса для оптимального обновления DOM (Document Object Model — «объектная модель документа») браузера. Таким образом программист может работать со страницей, считая, что она обновляется вся, но библиотека самостоятельно решает, какие компоненты страницы необходимо обновить. JavaScript XML (JSX) — это расширение синтаксиса JavaScript, которое позволяет использовать HTML-подобный синтаксис для описания структуры интерфейса. Как правило, компоненты написаны с использованием JSX, но также есть возможность использования обычного JavaScript.

3) Back-end часть разрабатываемого приложения строится по следующей архитектуре:

* Контроллеры – представляют из себя фасады для обращения к методам-сервисам, обработку ошибок и возвращения необходимых ответов от клиента. Сами контроллеры являются REST API контроллерами.
* DTO – представляет из себя слой представления, реализуемый через классы, в которых хранится информация об объекте из БД.
* Мапперы – интерфейсы для преобразования моделей в DTO.
* Модели – классы-сущности, являющимися таблицами в СУБД.
* Репозитории – ­интерфейсы и классы для реализации связи между моделями и СУБД непосредственно. Наследуются от JpaRepository.
* Сервисы – методы, реализующие всю логику приложения, активно взаимодействующие со всем выше обозначенными типами объектов.

Front-end часть разрабатываемого приложения строится по следующей архитектуре:

* Компоненты – части веб-страницы, которые реализуются отдельным модулями, для дальнейшего повторного использования кода. Модули реализуют в себе отображение и простую логику.
* Константы – классы, хранящие статические константы, такие как пути API приложения, URL пути приложения и др.
* Хуки – кастомизированные хуки.
* Модели – классы, в которых хранятся данные, получаемые после обращения к API. Являются аналогами DTO в back-end части.
* Страницы – страницы веб-приложения.
* Роутер – логика рутинга в веб-приложении.
* Сервисы – методы, реализующие всю сложную логику приложения, в том числе CRUD сервис для взаимодействия с API.
* Стили – CSS стили приложения.

4) Был реализован прототип системы.

В back-end части приложения были реализованы:

* Контроллер управления командами – представляет из себя конструктор и 3 метода: получения списка команд, заведение новой команды, удаление имеющейся команды.

Листинг 1 – Контроллер управления командами

@RestController

@CrossOrigin(origins = "http://localhost:3000", allowedHeaders = "\*", exposedHeaders = "\*")

@RequestMapping("/api/teams")

public class TeamsController {

private final ITeamsService \_teamsService;

public TeamsController (ITeamsService teamsService) {

\_teamsService = teamsService;

}

@GetMapping

public Iterable<TeamsDto> getAllTeams() {

return \_teamsService.getAllTeams();

}

@PostMapping()

public ResponseEntity<Teams> addNewTeam(@RequestBody Map<String, Object> dto) {

try {

Teams teams = \_teamsService.AddNewTeam(dto);

return new ResponseEntity<Teams>(HttpStatus.OK);

} catch (Error e) {

return new ResponseEntity<Teams>(HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

@DeleteMapping("/{id}")

public ResponseEntity deleteTeamById(@PathVariable Integer id) {

try {

\_teamsService.DeleteTeamById(id);

return new ResponseEntity<>(HttpStatus.OK);

} catch (EmptyResultDataAccessException e) {

ErrorApiResponse error = new ErrorApiResponse();

error.setErrorMsg(e.getMessage());

return new ResponseEntity<>(error, HttpStatus.NOT\_FOUND);

}

}

}

* Объект «команда» – обозначение атрибутов класса «команда», а также геттеры и сеттеры.

Листинг 2 – Объект «команда»

@Entity

public class Teams {

public Teams() {

}

public Teams(String name, Date dateTimeAdd, String shortName) {

Name = name;

DateTimeAdd = dateTimeAdd;

ShortName = shortName;

}

@Id

@GeneratedValue(strategy=GenerationType.AUTO)

private Integer Id;

private String Name;

private Date DateTimeAdd;

private String ShortName;

@Nullable

private byte[] Img;

@OneToMany(targetEntity = Players.class, cascade = CascadeType.ALL)

@JoinColumn(name = "team", referencedColumnName = "Id")

private List<Players> Players;

public List<com.example.footballers.models.Players> getPlayers() {

return Players;

}

public void setPlayers(List<com.example.footballers.models.Players> players) {

Players = players;

}

public String getShortName() {

return ShortName;

}

public void setShortName(String shortName) {

ShortName = shortName;

}

public Integer getId() {

return Id;

}

public void setId(Integer id) {

Id = id;

}

public String getName() {

return Name;

}

public void setName(String name) {

Name = name;

}

public Date getDateTimeAdd() {

return DateTimeAdd;

}

public void setDateTimeAdd(Date dateTimeAdd) {

DateTimeAdd = dateTimeAdd;

}

public byte[] getImg() {

return Img;

}

public void setImg(byte[] img) {

Img = img;

}

}

* Интерфейс сервиса управления командами – обозначение методов, которые будут реализованы в сервисе управления командами.

Листинг 3 – Интерфейс сервиса управления командами

public interface ITeamsService {

Iterable<TeamsDto> getAllTeams();

Teams AddNewTeam(Map<String, Object> dto);

void DeleteTeamById(Integer id);

}

* Сервис управления командами – реализация методов управления командами.

Листинг 4 – Сервис управления командами

@Service

public class TeamsService implements ITeamsService {

private final ITeamsRepos \_teamsRepos;

public TeamsService(ITeamsRepos teamsRepos) {

this.\_teamsRepos = teamsRepos;

}

public Iterable<TeamsDto> getAllTeams() {

return TeamsMapper.toDtoIterable(\_teamsRepos.findAll());

}

public Teams AddNewTeam(Map<String, Object> dto) {

Teams team = TeamsMapper.toModel(dto);

return \_teamsRepos.save(team);

}

public void DeleteTeamById(Integer id) {

\_teamsRepos.deleteById(id);

return;

}

}

В front-end части приложения были реализованы:

* Страница отображения команд.

Листинг 5 – Страница отображения команд

const TeamsPage = () => {

const [showModal, setShowModal] = useState(false);

const [error, setError] = useState(null);

useEffect( () => {

if (error === '') {

toast.success('Команда успешно добавлена', {

position: "top-center",

autoClose: 5000,

hideProgressBar: false,

closeOnClick: true,

pauseOnHover: true,

draggable: true,

progress: undefined,

theme: "light",

});

}

if (error !== '' && error !== null) {

toast.error(error, {

position: "top-center",

autoClose: 5000,

hideProgressBar: false,

closeOnClick: true,

pauseOnHover: true,

draggable: true,

progress: undefined,

theme: "light",

});

}

setError(null);

}, [error]);

return(

<div>

<div className="team\_\_div\_\_center">

<h1>Доступные команды</h1>

<Button variant="success" onClick={() => setShowModal(true)}><Icon.PlusCircle/> Добавить новую команду</Button>

</div>

<div>

<TeamsList render={showModal}/>

</div>

<TeamsDialog active={showModal} setActive={setShowModal} error={error} setError={setError}/>

</div>

);

}

export default TeamsPage;

* Crud Service.

Листинг 6 - Crud Service

export default class CrudService {

static async getAll(url) {

const response = await axios.get(String(url));

return response;

}

static async getAllByTeam(url) {

const response = await axios.get(String(url));

return response;

}

static async add(url, obj) {

const response = await axios.post(String(url), obj, {

headers: ApiPath.Headers

});

return response;

}

static async deleteById(url, id) {

const response = await axios.delete(String(url) + `/${id}`);

return response;

}

static async update(url, obj) {

const response = await axios.put((String)(url), obj, {

headers: ApiPath.Headers

});

return response;

}

}

5) Пример HTML страниц.

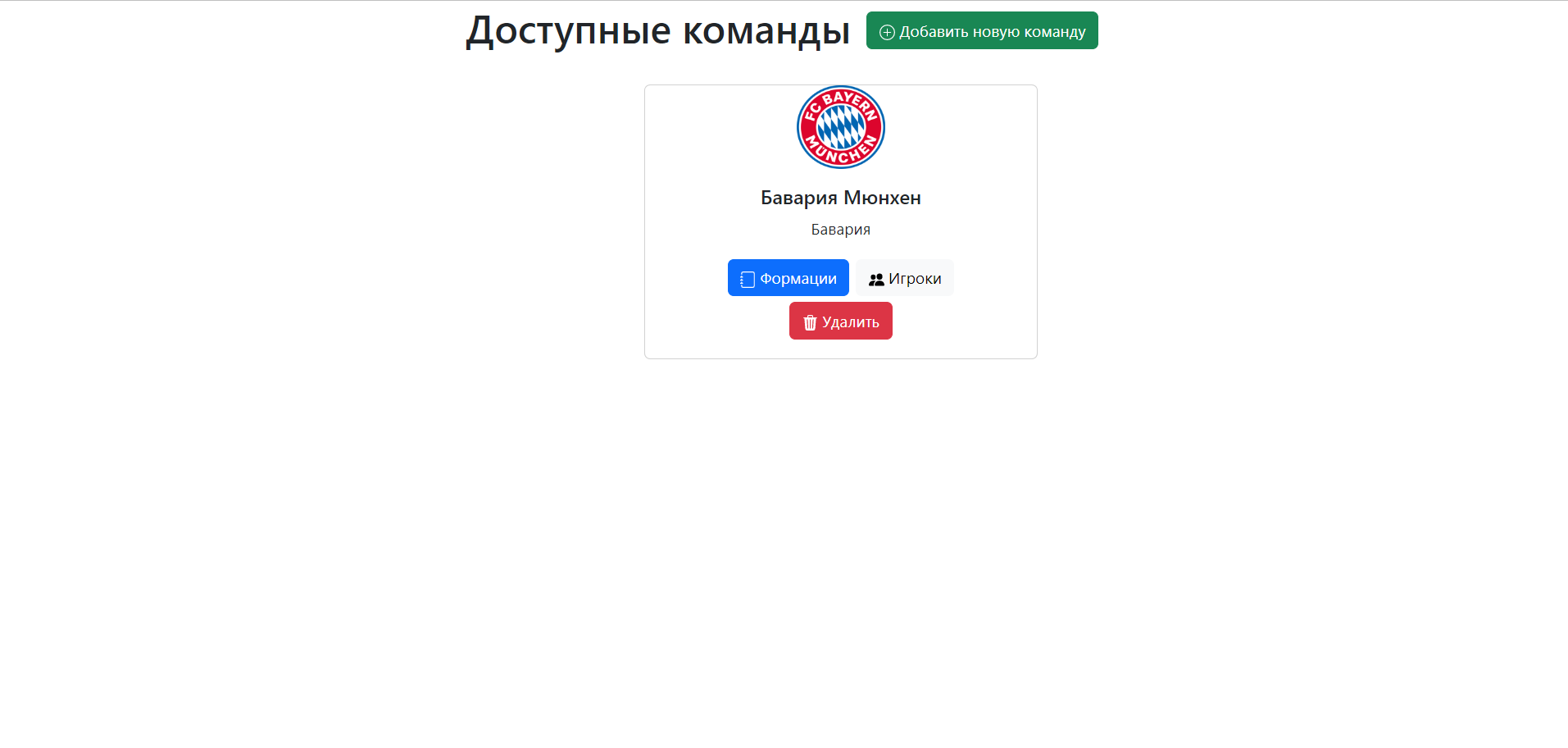


Рисунок 2 – Страница со списком доступных команд

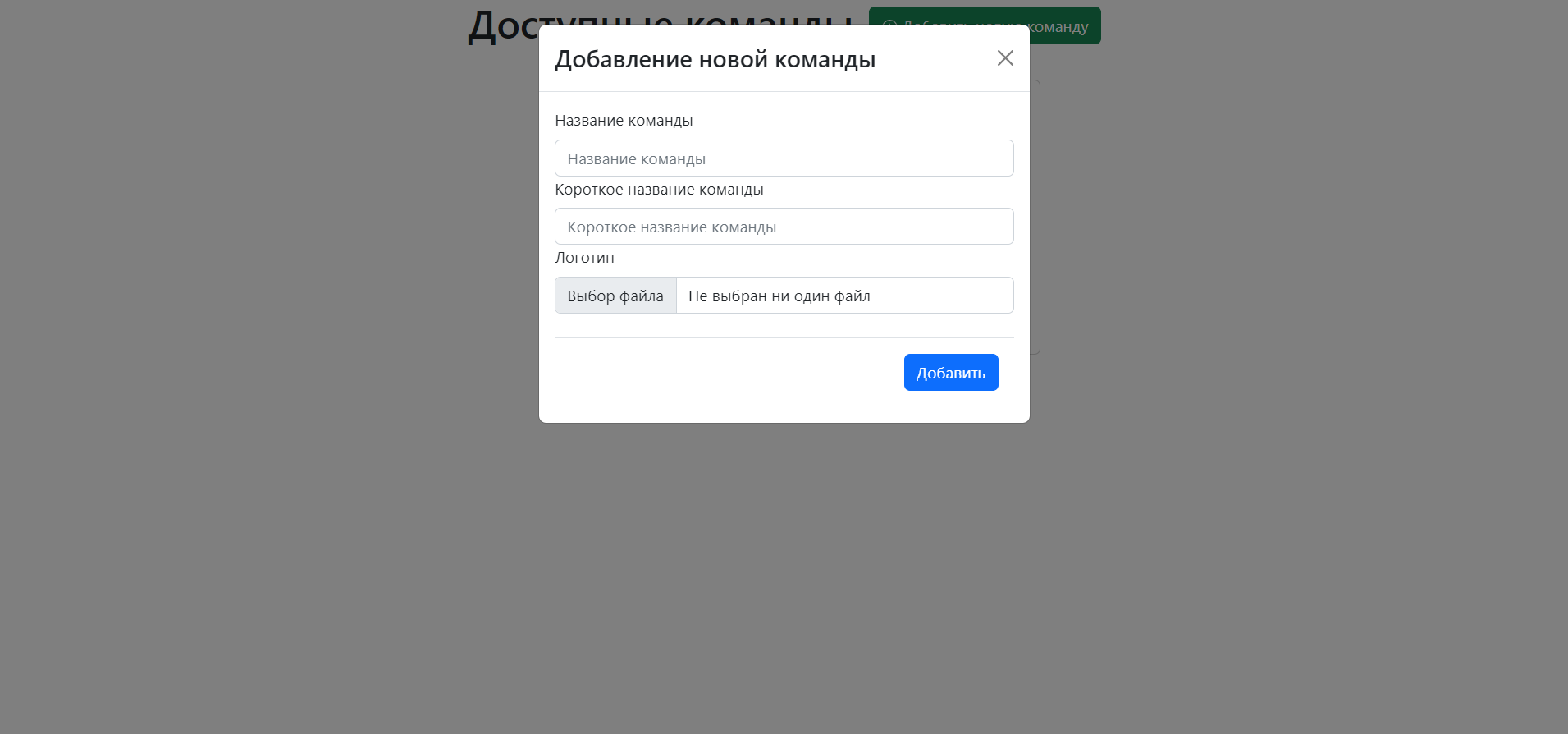


Рисунок 3 – Диалоговое окно добавления новой команды

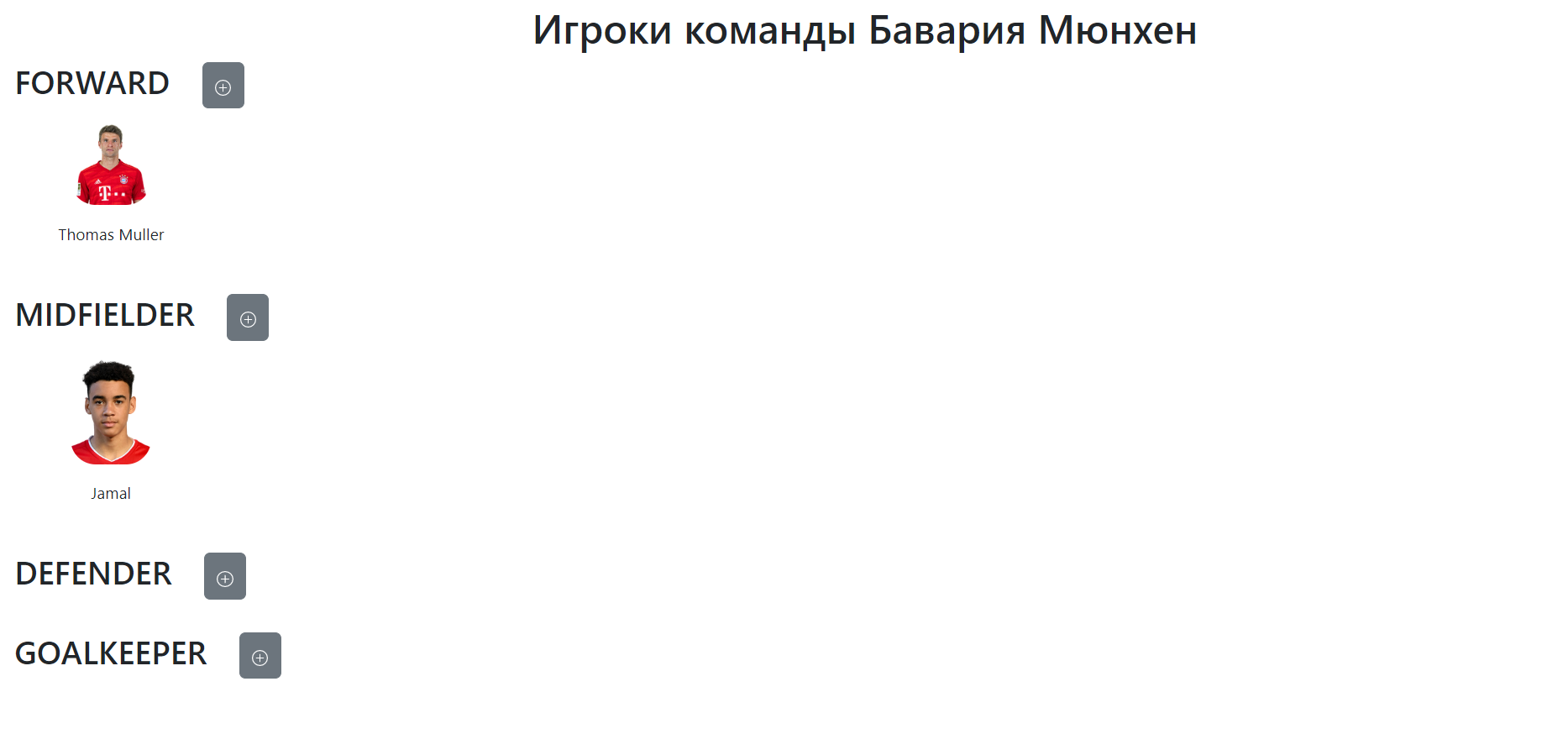


Рисунок 4 – Список игроков в команде

**Вывод:**

Была реализована автоматизированная система управления футбольными командами. Для back-end части был выбран Java Spring Framework, для front-end был выбран ReactJS. Была составлена архитектура приложения.