МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВАЯ РАБОТА   
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | А. В. Фомин |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОЙ РАБОТЕ | | | | | |
| РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УПРАВЛЕНИЯ ФУТБОЛЬНЫМИ СОСТАВАМИ | | | | | |
| по дисциплине: СОВРЕМЕННЫЕ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ | | | | | |
|  | | | | | |
|  | | | | | |

РАБОТУ ВЫПОЛНИЛ

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

Санкт-Петербург 2023

СОДЕРЖАНИЕ

[РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УПРАВЛЕНИЯ ФУТБОЛЬНЫМИ СОСТАВАМИ 1](#_Toc149253420)

[Введение 3](#_Toc149253421)

[1 Описание архитектуры разрабатываемого приложения 5](#_Toc149253422)

[1.1 Контекстный уровень архитектуры 5](#_Toc149253423)

[1.2 Контейнерный уровень архитектуры 6](#_Toc149253424)

[2 Обоснование выбора технических средств 11](#_Toc149253425)

[3 Описание структур данных 15](#_Toc149253426)

[4 Описание основных алгоритмов работы и схем взаимодействия клиентский и серверных скриптов web-приложения. 18](#_Toc149253427)

# Введение

Для организации слаженной и успешной работы тренерского персонала футбольного клуба необходимо иметь всеобъемлющую и полноценную систему, которая может покрывать задачи создания и хранения списка возможных составов команды или даже клуба в целом, включая в себя работу с нескольким командами разных возрастов. Для реализации данной системы необходимо преодолеть этапы анализа, проектирования и конечного составления системы. Выполнение данного задания позволит выработать навыки, необходимые для построения комплексных информационных систем.

В качестве основных функциональных требований к разрабатываемой системе можно выделить возможность создавать и вести учёт нескольких команд, к каждой команде может быть привязано несколько игроков, которые имеют чёткое разделение на позицию на поле. На основании списка игроков основной функцией считается создание составов. К нефункциональным требованиям можно отнести одновременное хранение команд не более 10 штук, список игроков для одной команды не более 50 человек, не более 15 штук составов для одной команды, а также простой и динамически адаптируемый интерфейс.

Для описания архитектуры, ER-модели и схем взаимодействия клиентской и серверной частей web-приложения будет использоваться PlantUML, который является достаточно комплексным и гибким инструментом для построения различного вида диаграмм.

**Цель:** проектирование и реализация комплексной информационной системы управления футбольными составами, которая призвана автоматизировать процесс создания различных составов в клубе.

**Задачи:**

* составить высокоуровневое описание архитектуры разрабатываемой информационной системы;
* выбрать технические средства для реализации информационной системы;
* составить описание структуры данных;
* описать и реализовать основные алгоритмы работы приложения;
* реализовать конечные интерфейсы системы.

# 1 Описание архитектуры разрабатываемого приложения

В качестве модели описания архитектуры были взяты первые два уровня модели C4. Данная модель является простым методом графического представления моделирования программных систем. Основанием модели является структурная декомпозиция системы на контейнеры и компоненты. Опирается на методы моделирования UML и ER. Модель состоит из 4-х уровней:

* Контекстный уровень – демонстрирует систему в масштабе её взаимодействия с пользователями и другими системами.
* Контейнерный уровень – разбивает систему на взаимосвязанные контейнеры, где контейнер это исполняемая и развертываемая система.
* Компонентный уровень – разделяет контейнеры на взаимосвязанные компоненты и отражают связь компонент с другими контейнерами или другими системами.
* Уровень кода - предоставляют дополнительные сведения о дизайне архитектурных элементов, которые могут быть сопоставлены с программным кодом.

Для описания архитектуры были использованы контекстный и контейнерный уровни.

# Контекстный уровень архитектуры

Контекстный уровень архитектуры, представленный на рисунке 1 и листинге 1, демонстрирует, что пользователь, в данном случае персонал тренерского штаба, посредством браузера обращается в разрабатываемую систему через интерактивной WEB-интерфейс и работает с системой через него, в это же время WEB-интерфейс обращается к API части системы, реализованному по принципу REST API. При необходимости черпать информацию из базы данных используется встроенный в фреймворк или внешний ORM-сервис.

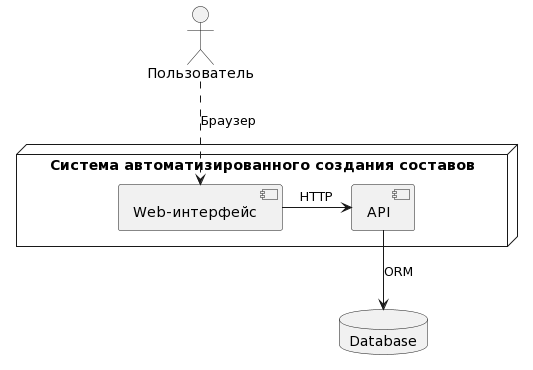


Рисунок 1 – Контекстный уровень архитектуры

Листинг 1 - Контекстный уровень архитектуры на языке PlantUML

@startuml

Пользователь as user

node "Система автоматизированного создания составов" {

[Web-интерфейс] as webInterface

[API] as api

}

database Database as db

user ..> webInterface: Браузер

webInterface -> api: HTTP

api --> db: ORM

@enduml

# Контейнерный уровень архитектуры

На контейнерном уровне рассматриваются WEB-интерфейс и API части системы.

Контейнерный уровень архитектуры WEB-интерфейса, который представлен на рисунке 2 и листинге 2, состоит из страниц, которые в свою очередь состоят из компонентов. И страницы, и компоненты отображаются с помощью классов стилей. Внутри каждого компонента присутствуют методы, которые обращаются к API для получения или отправки данных с помощью CRUD-сервисов.

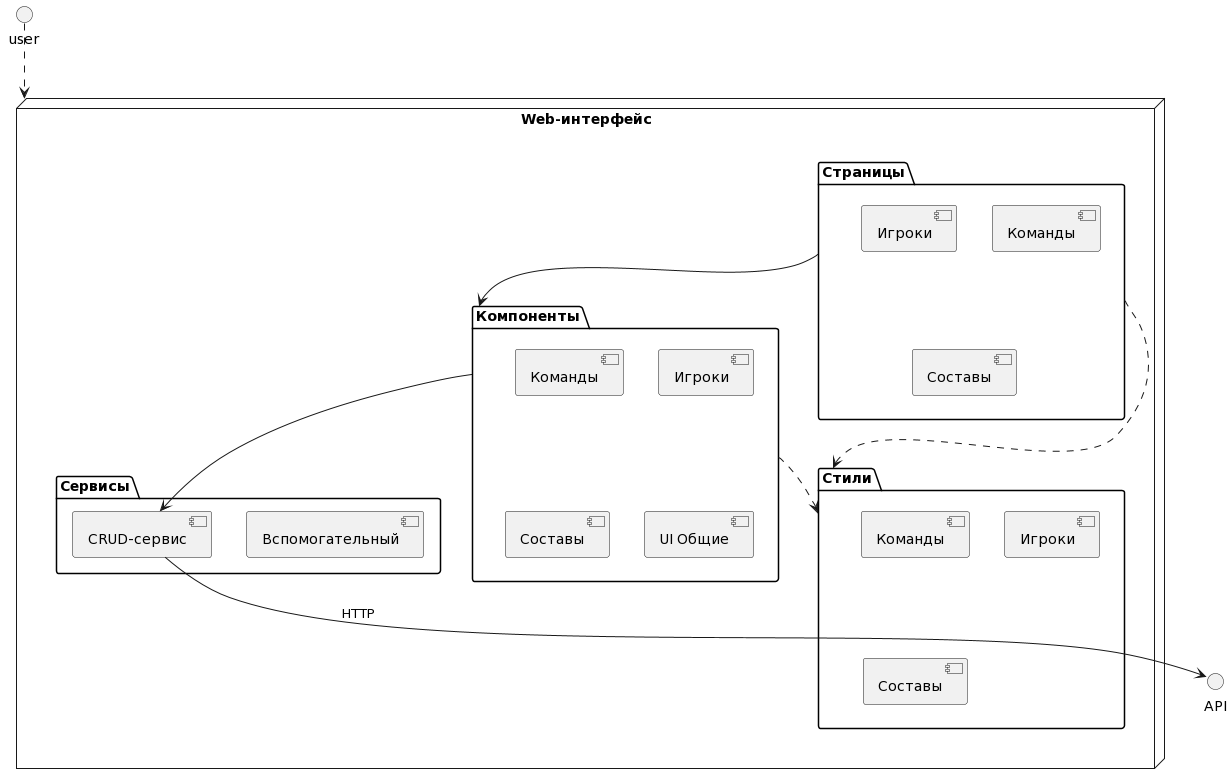


Рисунок 2 - Контейнерный уровень архитектуры WEB-интерфейса

Листинг 2 – Контейнерный уровень архитектуры WEB-интерфейса на языке PlantUML­­

@startuml

node "Web-интерфейс" as webInterface {

package Страницы as pages {

[Команды] as teamsPage

[Игроки] as playersPage

[Составы] as formationsPage

}

package Компоненты as components {

[Команды] as teamsComponent

[Игроки] as playersComponent

[Составы] as formationsComponent

[UI Общие] as uiComponent

}

package Стили as styles {

[Команды] as teamsStyles

[Игроки] as playersStyles

[Составы] as formationsStyles

}

package Сервисы as services {

[Вспомогательный] as utilsService

[CRUD-сервис] as crudService

}

}

user ..> webInterface

pages --> components

pages ..> styles

components ..> styles

components --> crudService

crudService --> API: HTTP

@enduml

Контейнерный уровень архитектуры API, который представлен на рисунке 3 и листинге 3, состоит из контроллеров, методы в котором соотносятся с URL адресами, по которым WEB-интерфейс обращается к API. Каждый метод в контроллере обращается к сервисам, которые декомпозируют запрос и несут в себе основные алгоритмы приложения. Для взаимодействия с базой данных сервисы обращаются к репозиториям и моделям через ORM.

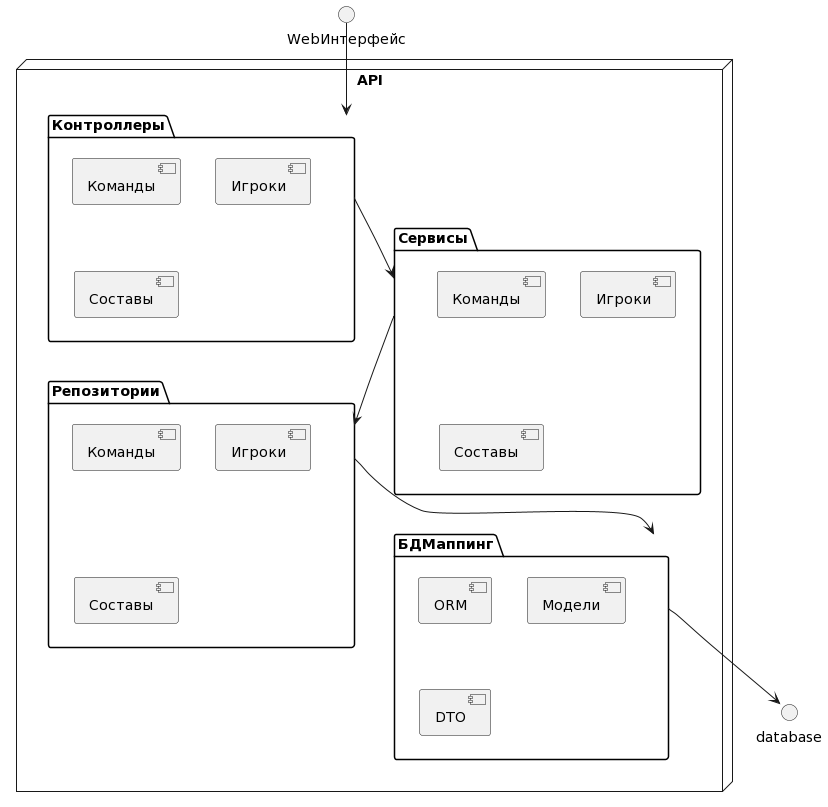


Рисунок 3 – Контейнерный уровень архитектуры WEB-интерфейса

Листинг 3 – Контейнерный уровень архитектуры WEB-интерфейса на языке PlantUML­­

@startuml

node "API" {

package Контроллеры as controllers {

[Команды] as teamsController

[Игроки] as playersController

[Составы] as formationsController

}

package Сервисы as services {

[Команды] as teamsServices

[Игроки] as playersServices

[Составы] as formationsServices

}

package Репозитории as repos {

[Команды] as teamsRepos

[Игроки] as playersRepos

[Составы] as formationsRepos

}

package БДМаппинг as dbMapping {

[ORM] as orm

[Модели] as models

[DTO] as dto

}

}

WebИнтерфейс --> controllers

controllers --> services

services --> repos

repos --> dbMapping

dbMapping --> database

@enduml

# 2 Обоснование выбора технических средств

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.

Spring Framework – является общим названием для ряда небольших фреймворков, каждый из которых выполняет свою работу. Структура Spring Framework представлена на рисунке 4.

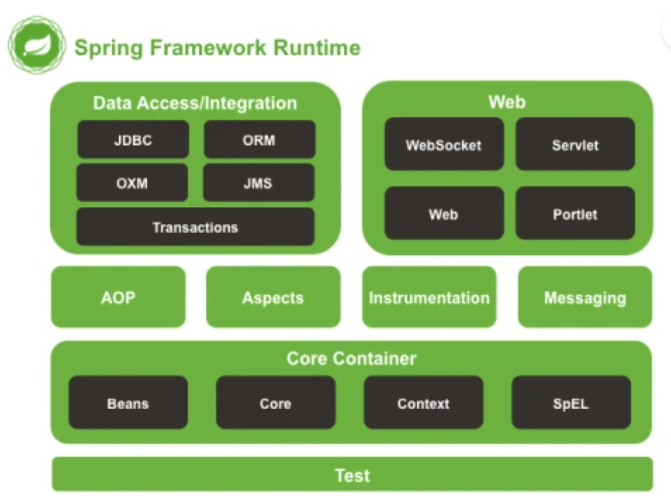


Рисунок 4 – Структура Java Spring Framework

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. Back-end часть разрабатываемого приложения строится по следующей структуре при использовании фреймворка Spring:

* 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.

Для back-end части нет необходимости использовать дополнительные библиотеки или инструменты по мимо стандартных библиотек, которые входят в Spring.

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. 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.

Front-end часть разрабатываемого приложения строится по следующей структуре:

* 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.

Важной частью front-end является подключение Bootstrap, который является свободным инструментом для создания сайтов и веб-приложений с точки зрения дизайна. Включает в себя огромное количество шаблонов оформления для типографики, веб-форм, кнопок, меток и прочих компонентов.

В качестве СУБД была выбрана MySQL. Данная СУБД отлично подходит для небольших WEB-приложений, коим и является данная разработка. Она обладает достаточной производительностью для одного клиента даже с большими объемами данных. Несмотря на то, что MySQL имеет ограниченный набор типов данных, их достаточно для реализовываемого приложения. Также, стоит обратить внимание, что полная версия СУБД доступна абсолютно бесплатно, так как она имеет открытый исходный код.

# 3 Описание структур данных

ER-модель базы данных системы представлена на рисунке 5 и листинге 4.

Таблица Teams содержит в себе информацию о каждой команде, по которой ведётся учёт в приложении. Таблица PositionList является справочной таблицей, в которой хранятся возможные позиции игроков. Стандартными позициями являются: FORWARD, MIDFIELDER, DEFENDER, GOALKEEPER. Таблица Players содержит в себе информацию о всех игроках, о которых ведётся учёт в приложении. Данная таблица включает в себя два внешних ключа к таблицам Teams и PositionList, которые определяют принадлежность игрока к определённой команде и его позицию соответственно.

Таблица Formations является справочной таблицей, в которой хранятся игровые схемы. Стандартными схемами являются: 4-3-3 и 4-2-3-1. Таблица GamePlans содержит в себе созданные составы команд. Данная таблица включает в себя два внешних ключа к таблицам Teams и Formations, которые определяют принадлежность состава к определённой команде и схему, на основе которой составлен состав. В поле Array типа json хранится массив с id игроков, которые задействованы в составе.

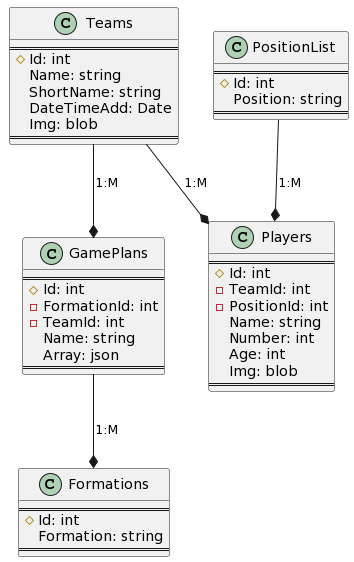


Рисунок 5 – ER-модель базы данных

Листинг 4 – ER-модель базы данных

@startuml

' UML legend:

' table = class

' #pkey

' -fkey

' +index

class Teams {

==

#Id: int

Name: string

ShortName: string

DateTimeAdd: Date

Img: blob

==

}

class Players {

==

#Id: int

-TeamId: int

-PositionId: int

Name: string

Number: int

Age: int

Img: blob

==

}

class PositionList {

==

#Id: int

Position: string

==

}

class Formations {

==

#Id: int

Formation: string

==

}

class GamePlans {

==

#Id: int

-FormationId: int

-TeamId: int

Name: string

Array: json

==

}

Teams --\* Players: 1:M

Teams --\* GamePlans: 1:M

PositionList --\* Players: 1:M

GamePlans --\* Formations: 1:M

@enduml

# 4 Описание основных алгоритмов работы и схем взаимодействия клиентский и серверных скриптов web-приложения.

Для описания основных алгоритмов работы и взаимодействия клиентских и серверных скриптов web-приложения будут реализованы диаграммы последовательности.

Алгоритмы и схемы взаимодействия клиентских и серверных скриптов добавления команд и игроков являются идентичными в системе. Суть алгоритма добавления игрока или команды заключается в запросе на добавление игрока или команды от пользователя, веб-интерфейс даёт возможность ввести данные об игроке или команде, после пользователь отправляет введённые данные, а веб-интерфейс обращается к API запросом типа POST. Внутри API сервис преобразует модель DTO к модели таблицы в базе данных, а затем отправляет преобразованную модель в базу. Затем база возвращает ответ API, в свою очередь API возвращается статус 200 в веб-интерфейс, а веб-интерфейс демонстрирует всплывающее окно об успешности добавления игрока или команды. Диаграммы последовательности добавления игрока или команды представлены на рисунках 6 и 7, а также в листингах 5 и 6 соответственно.

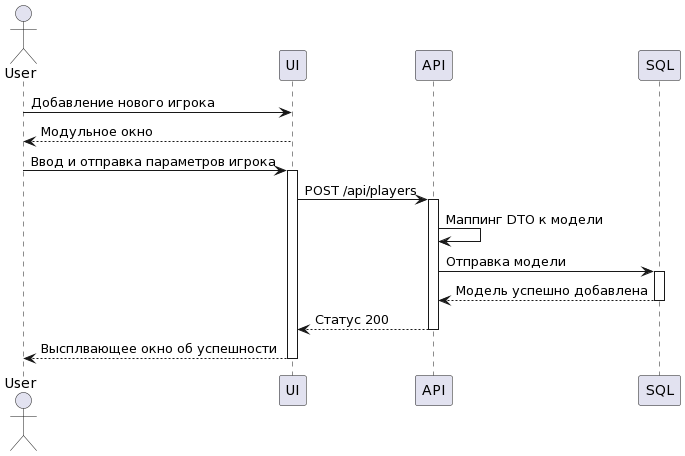


Рисунок 6 – Диаграмма последовательности добавления игрока

Листинг 5 – Диаграмма последовательности добавления игрока на языке PlantUML

@startuml

'seq diag for players (add)

actor User as user

participant "UI" as ui

participant "API" as api

participant "SQL" as sql

user -> ui: Добавление нового игрока

ui --> user: Модульное окно

user -> ui: Ввод и отправка параметров игрока

activate ui

ui -> api: POST /api/players

activate api

api -> api: Маппинг DTO к модели

api -> sql: Отправка модели

activate sql

sql --> api: Модель успешно добавлена

deactivate sql

api --> ui: Статус 200

deactivate api

ui --> user: Высплвающее окно об успешности

deactivate ui

@enduml

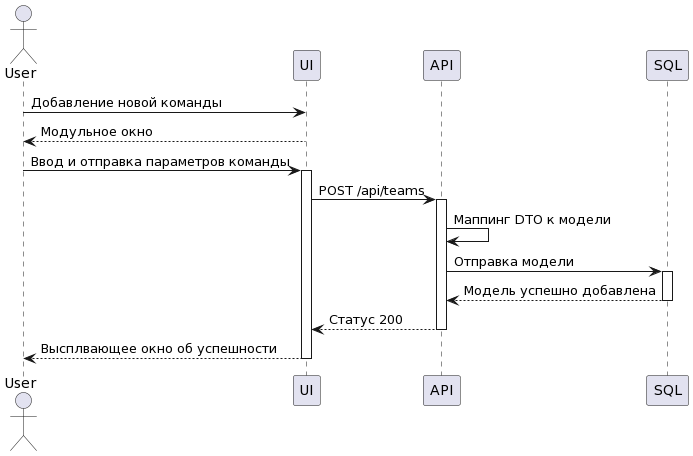


Рисунок 7 – Диаграмма последовательности добавления команды

Листинг 6 – Диаграмма последовательности добавления команды на языке PlantUML

@startuml

'seq diag for teams (add)

actor User as user

participant "UI" as ui

participant "API" as api

participant "SQL" as sql

user -> ui: Добавление новой команды

ui --> user: Модульное окно

user -> ui: Ввод и отправка параметров команды

activate ui

ui -> api: POST /api/teams

activate api

api -> api: Маппинг DTO к модели

api -> sql: Отправка модели

activate sql

sql --> api: Модель успешно добавлена

deactivate sql

api --> ui: Статус 200

deactivate api

ui --> user: Высплвающее окно об успешности

deactivate ui

@enduml

Алгоритмы и схемы взаимодействия клиентских и серверных скриптов удаления команд, игроков и составов являются идентичными в системе. Суть алгоритма добавления игрока, команды или состава заключается в запросе на удаления игрока или команды от пользователя, веб-интерфейс демонстрирует предупреждающе окно, после пользователь подтверждает удаление, а веб-интерфейс обращается к API запросом типа DELETE. Внутри API сервис отправляет запрос на удаление в зависимости от id. Затем база возвращает ответ API, в свою очередь API возвращается статус 200 в веб-интерфейс, а веб-интерфейс демонстрирует всплывающее окно об успешности удаления игрока, команды или состава. Диаграммы последовательности удаления игрока, команды или состава представлены на рисунках 8, 9, 10, а также в листингах 7, 8 и 9 соответственно.

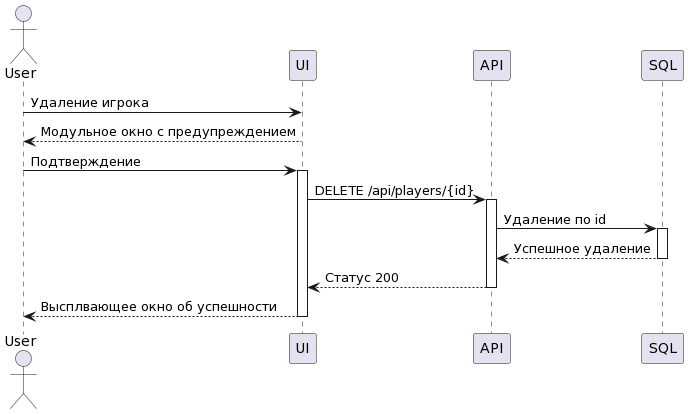


Рисунок 8 – Диаграмма последовательности удаления игрока

Листинг 7 – Диаграмма последовательности удаления игрока на языке PlantUML

@startuml

'seq diag for players (del)

actor User as user

participant "UI" as ui

participant "API" as api

participant "SQL" as sql

user -> ui: Удаление игрока

ui --> user: Модульное окно с предупреждением

user -> ui: Подтверждение

activate ui

ui -> api: DELETE /api/players/{id}

activate api

api -> sql: Удаление по id

activate sql

sql --> api: Успешное удаление

deactivate sql

api --> ui: Статус 200

deactivate api

ui --> user: Высплвающее окно об успешности

deactivate ui

@enduml

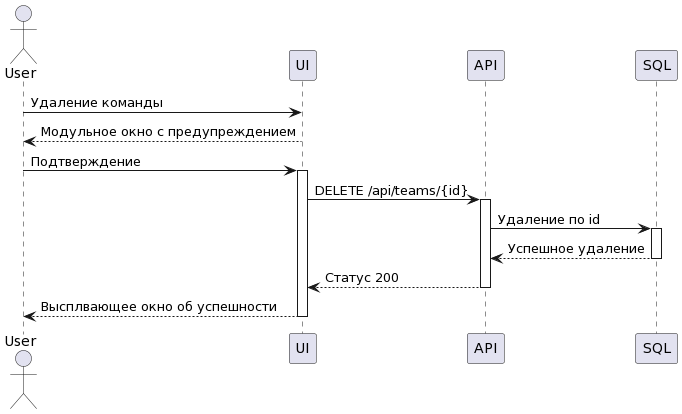


Рисунок 9 – Диаграмма последовательности удаления команды

Листинг 8 – Диаграмма последовательности удаления команды на языке PlantUML

@startuml

'seq diag for teams (del)

actor User as user

participant "UI" as ui

participant "API" as api

participant "SQL" as sql

user -> ui: Удаление команды

ui --> user: Модульное окно с предупреждением

user -> ui: Подтверждение

activate ui

ui -> api: DELETE /api/teams/{id}

activate api

api -> sql: Удаление по id

activate sql

sql --> api: Успешное удаление

deactivate sql

api --> ui: Статус 200

deactivate api

ui --> user: Высплвающее окно об успешности

deactivate ui

@enduml

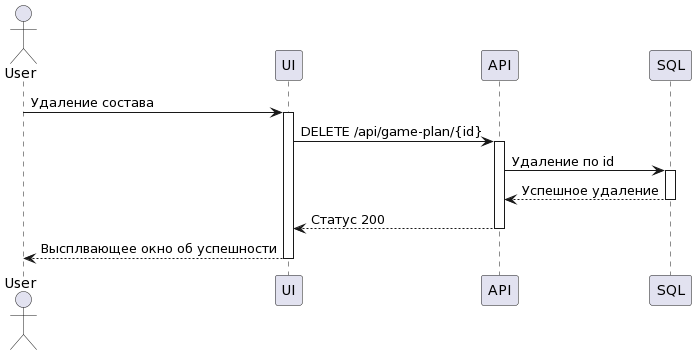


Рисунок 10 – Диаграмма последовательности удаления состава

Листинг 9 – Диаграмма последовательности удаления состава на языке PlantUML

@startuml

'seq diag for gamePlans (del)

actor User as user

participant "UI" as ui

participant "API" as api

participant "SQL" as sql

user -> ui: Удаление состава

activate ui

ui -> api: DELETE /api/game-plan/{id}

activate api

api -> sql: Удаление по id

activate sql

sql --> api: Успешное удаление

deactivate sql

api --> ui: Статус 200

deactivate api

ui --> user: Высплвающее окно об успешности

deactivate ui

@enduml

Алгоритм и схема взаимодействия клиентских и серверных скриптов изменения данных о игроке является схожей со схемами добавления нового игрока, однако вместо запроса типа POST, отправляется запрос типа PUT. Диаграмма последовательности изменения данных игрока представлена на рисунке 11, а также в листинге 10.

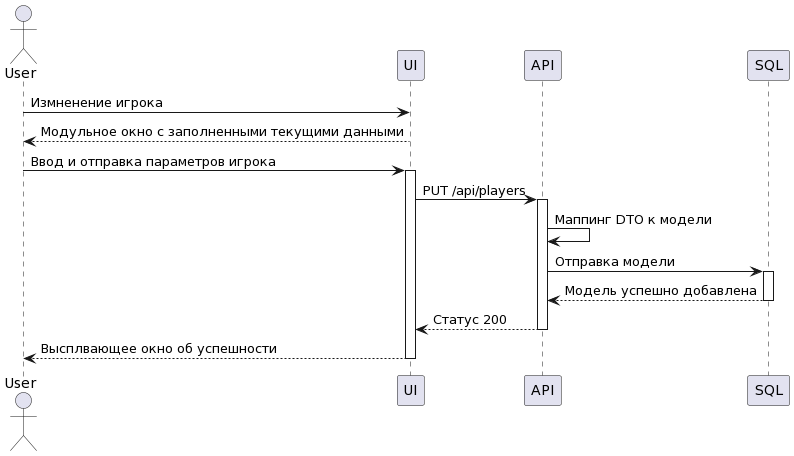


Рисунок 11 – Диаграмма последовательности изменения данных игрока

Листинг 10 – Диаграмма последовательности изменения данных игрока на языке PlantUML

@startuml

'seq diag for players (put)

actor User as user

participant "UI" as ui

participant "API" as api

participant "SQL" as sql

user -> ui: Измненение игрока

ui --> user: Модульное окно c заполненными текущими данными

user -> ui: Ввод и отправка параметров игрока

activate ui

ui -> api: PUT /api/players

activate api

api -> api: Маппинг DTO к модели

api -> sql: Отправка модели

activate sql

sql --> api: Модель успешно добавлена

deactivate sql

api --> ui: Статус 200

deactivate api

ui --> user: Высплвающее окно об успешности

deactivate ui

@enduml

Суть алгоритма и схемы взаимодействия клиентских и серверных скриптов добавления нового состава заключается в переходе на новую страницу создания состава, веб-интерфейс отправляется GET запрос в API на получения доступных схем. API обращается в базу данных за списком всех схем, а после получения ответа преобразует их в DTO модель и отправляет её в веб-интерфейс со статусом 200. Веб-интерфейс наполняет выпадающий список доступных схем. Пользователь выбирает схему, веб-интерфейс с помощью внутреннего сервиса генерирует интерактивное поле в зависимости от вида схемы. Пользователь выбирает позицию на поле, в которую он хочет поместить игрока, веб-интерфейс отправляет GET запрос со списком игроков по их позициям в API в зависимости от команды, API запрашивает список от базы данных, затем переводит их в DTO и отправляет ответ на веб-интерфейс со статусом 200 и преобразованной моделью DTO. Веб-интерфейс демонстрирует модальное окно со списком доступных игроков. Пользователь выбирает игрока, веб-интерфейс скрывает модальное окно и обновляет интерактивное поле. Пользователь повторяет действия выбора позиции и игрока на позицию. После вводит название состава, и нажимает на кнопку сохранения состава. Веб-интерфейс отправляет POST запрос с моделью в API, API преобразует DTO модель в модель таблицы базы данных, отправляет запрос в базу на сохранение, а после ответа от базы данных отправляет статус 200 в веб-интерфейс, а веб-интерфейс переводит пользователя на просмотр нового состава, а также демонстрирует всплывающее окно с информацией об успешности сохранения состава.