Dokumentace k projektu 3.6

Obsah

[1 Použité technologie 3](#_Toc135904791)

[1.1 FrontEnd 3](#_Toc135904792)

[1.2 BackEnd 3](#_Toc135904793)

[1.3 Database 3](#_Toc135904794)

[1.4 Push notifikace 3](#_Toc135904795)

[2 Vypracování jednotlivých komponent 3](#_Toc135904796)

[2.1 Server 3](#_Toc135904797)

[2.1.1 Seznam EndPointu 3](#_Toc135904798)

[2.1.2 Nastevení Push notifikací 4](#_Toc135904799)

[2.2 Database 4](#_Toc135904800)

[2.3 Administrátorská aplikace 4](#_Toc135904801)

[2.4 Objednávací panel, panel V Přípravě, panel Výdej 5](#_Toc135904802)

# Použité technologie

Aplikace se skládá z několika částí, počínaje webovým rozhraním a konče db.

## FrontEnd

FrontEnd je vystavěný na serveru Node.js a využívá knihovny React a několik dalších závislostí. Běží na portu 3000 a zpřístupňuje jak panel pro objednávání, tak panel v přípravě a panel výdej.

## BackEnd

BackEnd je napsán v Java 17 a je vytaven na frameworku Spring. Poskytuje endpointy pro práci s daty, která jsou dostupné na portu 8080.

## Database

Jako databázi jsem použil MongoDB.

## Push notifikace

Push notifikace jsou řešené protokolem WebSocket a Stomp.

# Vypracování jednotlivých komponent

Celý systém se skládá z několika komponent. Středobodem se server poskytující REST FULL api. Většina dalších komponent má podobu webového GUI.

## Server

Server je středobod celého systému. Poskytuje REST FULL api přes protokol http na portu 8080. Server disponuje endpointy pro websocket protokol a stomp.

Endpointy zpracovávají třídy xxControllet. Kontrolery pro práci s daty používají třídy xxService. Rozhraní xxRepository jen deklarují metody pro načítání z db.

Příklad

OrderController <-Order-> OrderService <-Order-> OrderRepository

### Seznam EndPointu

? – orders, categories, improvements, products

Get all - /api/?/ (GET)

Get one - /api/?/id (GET)

Add one - /api/?/ (POST)

Update one - /api/?/id (PUT)

Delete one - /api/?/id (DELETE)

Get all (deleted) - /api/?/data\_recovery/ (GET)

Requesty vrací status - 200 pokud prošel úspěšně

* 400 pokud data nejsou validní (přidání a update), nebo neexistují (získání, mazání)
* 500 chyba za kterou přijímá server zodpovědnost

Server uchovává data o Objednávkách, Kategoriích, Produktech a jejich Vylepšeních.

### Nastevení Push notifikací

Konfigurace websocketu

@Configuration

@EnableWebSocketMessageBroker

public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

  @Override

  public void configureMessageBroker(MessageBrokerRegistry config) {

    config.enableSimpleBroker("/orders");

    config.setApplicationDestinationPrefixes("/app");

  }

  @Override

  public void registerStompEndpoints(StompEndpointRegistry registry) {

    registry.addEndpoint("/ws").setAllowedOriginPatterns("\*");

    registry.addEndpoint("/ws").setAllowedOriginPatterns("\*").withSockJS();

  }

}

Controller, který mapuje pod /app/distributeOrder metodu, která přepošle order všem odběratelům /order/distribujeorder jako ve formátu json

@RestController

public class WebSocketController {

    @MessageMapping("/distributeOrder")

    @SendTo("/orders/distributeOrder")

    public Order distributeOrder(Order order) {

        return order;

    }

}

Odeslání zprávy přes stomp

    public void sendOrder(Order order) {

        session.send("/app/distributeOrder", order);

    }

## Database

Server používá pro uchování dat MongoDB.

## Administrátorská aplikace

Je v podstatě jeden předešlích projektů se dvěma výjimkami: pro získávání dat nepoužívá RMI registry, ale REST api serveru přes protokol HTTP a pro zprovoznění Push notifikací využívá protokoly websocket a stomp.

Spuštění websocketu na klientu

        // připojení přes websocket na server

        WebSocketClient client = new StandardWebSocketClient();

        WebSocketStompClient stompClient = new WebSocketStompClient(client);

        stompClient.setMessageConverter(new MappingJackson2MessageConverter());

        OrderStompSessionHandler sessionHandler = new OrderStompSessionHandler();

        stompClient.connectAsync("ws://localhost:8080/ws", sessionHandler);

Nastavení získávání dat ze serveru

        // nastevení získávání dat ze servru

        RestApi restApi = new RestApi("http://localhost:8080/api/");

        mountEndpoints(restApi);

        DB db = new DB(restApi);

Konstruktor pro RestApi chce url na server do url je možné přidat i cestu k rest api. Třída slouží k získávání dat z rest api.

Třída RestApi disponuje generickými metodami:

    public <E> List<E> getAll(Class<E> type) throws EndpointNotFound, IOException, JsonMappingException {

    public <E> E get(Class<E> type, String id) throws EndpointNotFound, IOException, JsonMappingException {

    public <E> E post(E data) throws EndpointNotFound, IOException, JsonMappingException {

    public <E> E put(E data, String id) throws EndpointNotFound, IOException, JsonMappingException {

    public <E> E delete(E data, String id) throws EndpointNotFound, IOException, JsonMappingException {

aby bylo možné používat tyto metody je třeba nastavit endpointy pomocí metody

public <E> void mountEndpoint(HTTP\_METHOD method, Class<E> type, String endpoint) throws MountingEndPointExeption {

HTTP\_METHOD je výčtový typ vnořený do třídy RestApi. Tento výčtový typ obsahuje metody HTTP. Endpoint se dá odebrat metodou

    public <E> void unmountEndpoint(HTTP\_METHOD method, Class<E> type)

V administrační aplikaci instanci RestApi obsluhuje třída DB.

## Objednávací panel, panel V Přípravě, panel Výdej

Ovládaví panely jsou provedeny formou webové aplikace.

Pro připojení k službě websocketu v reacku je použita závislost ‚@stomp/stompjs‘

import { Stomp } from '@stomp/stompjs'

const url = "ws://localhost:8080/ws"

const client = Stomp.client(url)

Všechny použité technologie jsou popsány v minulých dokumentacích