**ПРИЛОЖЕНИЕ 3**

**ПРОГРАММНЫЙ МОДУЛЬ ВИЗУАЛИЗАЦИИ ДЛЯ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ УПРАВЛЕНИЯ НАСОСНОЙ СТАНЦИИ (ПМ АСУНС)**

**ТЕКСТ ПРОГРАММЫ**

Москва, 2020

**Содержание**

[1. Главная функция backend сервера 3](#_Toc43578330)

[2. Генератор данных показателей датчиков станции 5](#_Toc43578331)

[3. Модуль аварийных ситуаций 10](#_Toc43578332)

[4. Компонент авторизации пользователя в системе 11](#_Toc43578333)

[5. Компонент главного экрана станции 16](#_Toc43578334)

# 1. Главная функция backend сервера

AuthController.java

package ru.asuns.controller;  
  
import lombok.extern.slf4j.Slf4j;  
import org.json.JSONObject;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import ru.asuns.exception.ResourceNotFoundException;  
import ru.asuns.model.User;  
import ru.asuns.payload.ApiResponse;  
import ru.asuns.payload.JwtAuthenticationResponse;  
import ru.asuns.payload.LoginRequest;  
import ru.asuns.payload.SignUpRequest;  
import ru.asuns.security.JwtTokenProvider;  
import ru.asuns.service.impl.AuthenticationService;  
import ru.asuns.service.impl.UserService;  
  
import javax.validation.Valid;  
import java.nio.file.attribute.UserPrincipalNotFoundException;  
import java.util.UUID;  
  
@Slf4j  
@RestController  
@RequestMapping("/auth")  
public class AuthController {  
  
 @Autowired  
 private UserService userService;  
  
 @Autowired  
 private AuthenticationService authenticationService;  
  
 @Autowired  
 private JwtTokenProvider tokenProvider;  
  
 @PostMapping("/signin")  
 public ResponseEntity authenticateUser(@Valid @RequestBody LoginRequest loginRequest) {  
 return authenticationService.authenticate(loginRequest);  
 }  
  
 @PostMapping("/signup")  
 public ResponseEntity<ApiResponse> registerUser(@RequestBody SignUpRequest signUpRequest) {  
 return authenticationService.registration(signUpRequest);  
 }  
  
 @GetMapping("/current/{token}")  
 public User getCurrentUser(@PathVariable String token) throws ResourceNotFoundException {  
 return userService.getUserById(tokenProvider.getUserIdFromJWT(token));  
 }  
  
 @GetMapping("/activation/{uuid}")  
 public ResponseEntity<ApiResponse> activation(@PathVariable UUID uuid) throws UserPrincipalNotFoundException {  
 return userService.activate(uuid);  
 }  
  
 @PostMapping("/validate")  
 public boolean validateToken(@RequestBody String token) {  
 JSONObject tokenJson = new JSONObject(token);  
 return tokenProvider.validateToken(tokenJson.get("value").toString());  
 }  
}

BaseStationController

package ru.asuns.controller;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;  
import ru.asuns.model.BaseStation;  
import ru.asuns.service.BaseStationService;  
  
import java.net.URI;  
import java.util.List;  
  
@RestController  
@RequestMapping("/station")  
public class BaseStationController {  
  
 @Autowired  
 BaseStationService baseStationService;  
  
 @GetMapping()  
 public @ResponseBody  
 List<BaseStation> getAllBaseStations() {  
 return baseStationService.getAllBaseStations();  
 }  
  
 @GetMapping("/active")  
 public @ResponseBody  
 List<BaseStation> getAllActiveBaseStations() {  
 return baseStationService.getAllActiveBaseStations();  
 }  
  
 @GetMapping("/{stationId}")  
 public @ResponseBody  
 BaseStation getStationById(@PathVariable Long stationId) {  
 return baseStationService.getBaseStationById(stationId);  
 }  
  
 @PostMapping("/{stationId}/state")  
 public @ResponseBody  
 boolean changeStationState(@PathVariable Long stationId) {  
 return baseStationService.changeStationState(stationId);  
 }  
  
 @PutMapping()  
 public @ResponseBody  
 ResponseEntity<BaseStation> createBaseStation(@RequestBody BaseStation station) {  
 BaseStation savedStation = baseStationService.createBaseStation(station);  
 URI uri = ServletUriComponentsBuilder.*fromCurrentContextPath*()  
 .path("/station/{stationId}")  
 .build(savedStation.getId());  
 return ResponseEntity.*created*(uri).build();  
 }  
  
 @DeleteMapping("/{stationId}")  
 public @ResponseBody  
 ResponseEntity createBaseStation(@PathVariable Long stationId) {  
 baseStationService.deleteBaseStation(stationId);  
 return ResponseEntity.*ok*().build();  
 }  
  
}

# 2. Генератор данных показателей датчиков станции

BaseStationRepository.java

package ru.asuns.config;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.boot.ApplicationArguments;  
import org.springframework.boot.ApplicationRunner;  
import org.springframework.stereotype.Component;  
import ru.asuns.model.\*;  
import ru.asuns.model.dto.UserDTO;  
import ru.asuns.repository.BaseStationRepository;  
import ru.asuns.repository.SensorRepository;  
import ru.asuns.repository.SensorValueTypeRepository;  
import ru.asuns.repository.UserRepository;  
  
import java.util.ArrayList;  
import java.util.List;  
  
@Component  
public class AppInit implements ApplicationRunner {  
  
 @Autowired  
 private SensorValueTypeRepository sensorValueTypeRepository;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private BaseStationRepository baseStationRepository;  
  
 @Autowired  
 private SensorRepository sensorRepository;  
  
 @Override  
 public void run(ApplicationArguments args) {  
  
 List<User> users = new ArrayList<>();  
 List<BaseStation> stations = new ArrayList<>();  
 List<Sensor> sensors = new ArrayList<>();  
  
 addValueType(types, "Ток НА1", "A");  
 addValueType(types, "Ток НА2", "A");  
 addValueType(types, "Ток НА3", "A");

addValueType(types, "Ток НА4", "A");  
 addValueType(types, "Ток НАлр", "А");  
 addValueType(types, "Поток F1", "MH");

addValueType(types, "Поток F2", "MH");  
 addValueType(types, "ТУ воды в ПО", "P");

addValueType(types, "Температура", "T");  
 addValueType(types, "Влажность", "H");  
 addValueType(types, "Освещение", "LM");

sensorValueTypeRepository.saveAll(types);  
  
 addUser(users, new UserDTO("admin", "admin", "123@mail.ru", Role.*ADMIN*));  
 addUser(users, new UserDTO("username","username", "456@mail.ru", Role.*USER*));  
  
 userRepository.saveAll(users);  
  
 addStation(stations, "ЗелАО", "11111");

baseStationRepository.saveAll(stations);  
  
 }  
  
   
 private void addValueType(List<SensorValueType> types, String name, String id) {  
 SensorValueType type = new SensorValueType(name, id);  
 if (!sensorValueTypeRepository.existsByCode(id))  
 types.add(type);  
 }  
  
 private void addUser(List<User> users, UserDTO userDTO) {  
 User user = new User(userDTO);  
 user.setActive(true);  
 if (!userRepository.existsUserByUsername(userDTO.getUsername()))  
 users.add(user);  
 }  
  
 private void addStation(List<BaseStation> stations, String name, String serialNumber) {  
 BaseStation station = new BaseStation(name, serialNumber);  
 if (!baseStationRepository.existsBySerialNumber(serialNumber))  
 stations.add(station);  
 }  
}

UserService.java

package ru.asuns.service.impl;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import ru.asuns.exception.ResourceNotFoundException;  
import ru.asuns.model.User;  
import ru.asuns.model.dto.UserDTO;  
import ru.asuns.payload.ApiResponse;  
import ru.asuns.repository.UserRepository;  
  
import java.nio.file.attribute.UserPrincipalNotFoundException;  
import java.util.UUID;  
  
@Service  
public class UserService implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 return userRepository.findUserByUsername(username)  
 .orElseThrow(() ->  
 new UsernameNotFoundException("User not found with username: " + username)  
 );  
 }  
  
 public ResponseEntity<ApiResponse> activate(UUID uuid) throws UserPrincipalNotFoundException {  
 User userData = userRepository.findByActivationCode(uuid).orElseThrow(  
 () -> new UserPrincipalNotFoundException("User by activation code not found")  
 );  
 userData.setActive(true);  
 userData.setActivationCode(null);  
 userRepository.save(userData);  
 return new ResponseEntity<>(new ApiResponse(true, "Email has been confirmed."), HttpStatus.*OK*);  
 }  
  
 public User getUserById(Long userId) throws ResourceNotFoundException {  
 return userRepository.findUserById(userId).orElseThrow(() -> new ResourceNotFoundException("User", "user\_id", userId));  
 }  
  
 public Long createUser(UserDTO userDTO) {  
 User user = new User(userDTO);  
 return userRepository.save(user).getId();  
 }  
  
 public void deleteUser(Long userId) throws ResourceNotFoundException {  
 User user = userRepository.findUserById(userId).orElseThrow(() -> new ResourceNotFoundException("User", "user\_id", userId));  
 userRepository.deleteById(userId);  
 }  
  
 private String encode(String password) {  
 return (new BCryptPasswordEncoder()).encode(password);  
 }  
}

AuthenticationService.java

package ru.asuns.service.impl;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.stereotype.Service;  
import org.springframework.web.servlet.support.ServletUriComponentsBuilder;  
import ru.asuns.exception.AppException;  
import ru.asuns.model.User;  
import ru.asuns.payload.ApiResponse;  
import ru.asuns.payload.JwtAuthenticationResponse;  
import ru.asuns.payload.LoginRequest;  
import ru.asuns.payload.SignUpRequest;  
import ru.asuns.repository.UserRepository;  
import ru.asuns.security.JwtTokenProvider;  
  
import java.net.URI;  
  
@Service  
public class AuthenticationService {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private JwtTokenProvider tokenProvider;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private MailSenderService senderService;  
  
 @Value("${server.real-ip}")  
 private String realIp;  
  
 public ResponseEntity authenticate(LoginRequest loginRequest) {  
  
 User candidate = userRepository.findUserByUsername(loginRequest.getUsernameOrEmail()).orElse(  
 userRepository.findUserByEmail(loginRequest.getUsernameOrEmail()).orElse(null)  
 );  
  
 if (candidate != null) {  
 Authentication authentication = authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(  
 candidate.getUsername(),  
 loginRequest.getPassword()  
 )  
 );  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
  
 User principal = (User) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();  
 String jwt = tokenProvider.generateToken(authentication);  
 return ResponseEntity.*ok*(new JwtAuthenticationResponse(jwt, principal));  
 }  
 return ResponseEntity.*badRequest*().body("No user with such credentials.");  
 }  
  
 public ResponseEntity<ApiResponse> registration(SignUpRequest signUpRequest) {  
 if (userRepository.existsUserByUsername(signUpRequest.getUsername())) {  
 return new ResponseEntity<>(new ApiResponse(false, "Username is already taken!"),  
 HttpStatus.*OK*);  
 }  
  
 if (userRepository.existsUserByEmail(signUpRequest.getEmail())) {  
 return new ResponseEntity<>(new ApiResponse(false, "Email Address already in use!"),  
 HttpStatus.*OK*);  
 }  
  
 User user = new User(signUpRequest);  
  
 try {  
 senderService.send(user.getEmail(), "Проект Сенсорика", "Ссылка для активации: http://" + realIp + "/api/auth/activation/" + user.getActivationCode());  
 } catch (Exception e) {  
 throw new AppException("Some problems with sending verification email. Check your email.");  
 }  
  
 User result = userRepository.save(user);  
  
 URI location = ServletUriComponentsBuilder  
 .*fromCurrentContextPath*().path("/users/{userId}")  
 .buildAndExpand(result.getId()).toUri();  
  
 return ResponseEntity.*created*(location).body(new ApiResponse(true, "User registered successfully"));  
 }  
}

# 3. Модуль аварийных ситуаций

# 

accident.ts

**import** {BaseStation} **from** "./base.station";  
  
**export class** Accident {  
 date: **number**;  
 reason: **string**;  
 causedBy: BaseStation;  
 criticalValue: **number**;  
 type: **string**;  
}

auth-guard.module.ts

**import** {Injectable} **from** '@angular/core';  
**import** {ActivatedRouteSnapshot, CanActivate, Router, RouterStateSnapshot} **from** '@angular/router';  
**import** {map} **from** 'rxjs/operators';  
**import** {AuthenticationService} **from** '../services/authentication.service';  
  
@Injectable({providedIn: 'root'})  
**export class** AuthGuardModule **implements** CanActivate {  
 **constructor**(  
 **private** router: Router,  
 **private** authenticationService: AuthenticationService  
 ) {  
 }  
  
 canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {  
 **const** currentAccessToken = **this**.authenticationService.currentAccessTokenValue;  
 **if** (currentAccessToken && currentAccessToken.active) {  
 **return true**;  
 }  
 **this**.router.navigate(['/login']);  
 **return false**;  
 }  
}

accident.component.spec.ts

**import** {async, ComponentFixture, TestBed} **from** '@angular/core/testing';  
  
**import** {AccidentComponent} **from** './accident.component';  
  
describe('AccidentComponent', () => {  
 **let** component: AccidentComponent;  
 **let** fixture: ComponentFixture<AccidentComponent>;  
  
 beforeEach(async(() => {  
 TestBed.configureTestingModule({  
 declarations: [AccidentComponent]  
 })  
 .compileComponents();  
 }));  
  
 beforeEach(() => {  
 fixture = TestBed.createComponent(AccidentComponent);  
 component = fixture.componentInstance;  
 fixture.detectChanges();  
 });  
  
 it('should create', () => {  
 expect(component).toBeTruthy();  
 });  
});

# 4. Компонент авторизации пользователя в системе

login.component.ts

**import** {Component, OnInit} **from** '@angular/core';  
**import** {ActivatedRoute, Router} **from** '@angular/router';  
**import** {FormBuilder, FormGroup, Validators} **from** '@angular/forms';  
**import** {first} **from** 'rxjs/operators';  
**import** {AuthenticationService} **from** '../../shared/services/authentication.service';  
**import** {SignUpRequest} **from** '../../shared/models/sign.up.request';  
  
@Component({  
 selector: 'app-login',  
 templateUrl: './login.component.html',  
 styleUrls: ['./login.component.css']  
})  
**export class** LoginComponent **implements** OnInit {  
 loginForm: FormGroup;  
 registrationForm: FormGroup;  
 returnUrl: **string**;  
 modalOpened: **boolean**;  
 signUpSuccess: **boolean**;  
 regErrorMessage: **string**;  
 regError: **boolean**;  
 loginError = **false**;  
 logErrorMessage: **string**;  
 proccedRegistration = **false**;  
  
 **constructor**(  
 **private** formBuilder: FormBuilder,  
 **private** route: ActivatedRoute,  
 **private** router: Router,  
 **private** authenticationService: AuthenticationService  
 ) {  
 **if** (**this**.authenticationService.currentAccessTokenValue) {  
 **this**.router.navigate(['/scheme-pumpstation']);  
 }  
 }  
  
 ngOnInit() {  
 **this**.loginForm = **this**.formBuilder.group({  
 usernameOrEmail: ['', [  
 Validators.*required*,  
 Validators.*minLength*(5)  
 ]],  
 password: ['', [  
 Validators.*required*,  
 Validators.*minLength*(8),  
 Validators.*pattern*(/^[a-zA-Z0-9!@#$%^&\*()\_]+$/)]  
 ]  
 });  
  
 **this**.registrationForm = **this**.formBuilder.group({  
 username: ['', [  
 Validators.*required*,  
 Validators.*minLength*(5)  
 ]],  
 password: ['', [  
 Validators.*required*,  
 Validators.*minLength*(8),  
 Validators.*pattern*(/^[a-zA-Z0-9!@#$%^&\*()\_]+$/)]  
 ],  
 email: ['', [  
 Validators.*required*,  
 Validators.*email* ]]  
 });  
  
 **this**.returnUrl = **this**.route.snapshot.queryParams.returnUrl || '/';  
 }  
  
 **get** loginControls() {  
 **return this**.loginForm.controls;  
 }  
  
 **get** registrationControls() {  
 **return this**.registrationForm.controls;  
 }  
  
 showError(msg: **string**) {  
 **this**.logErrorMessage = msg;  
 **this**.loginError = **true**;  
 setTimeout(() => {  
 **this**.loginError = **false**;  
 }, 5000);  
 }  
  
 showSuccess(msg: **string**) {  
 **this**.logErrorMessage = msg;  
 **this**.signUpSuccess = **true**;  
 setTimeout(() => {  
 **this**.signUpSuccess = **false**;  
 }, 5000);  
 }  
  
 registrationError(msg: **string**) {  
 **this**.regErrorMessage = msg;  
 **this**.regError = **true**;  
 setTimeout(() => {  
 **this**.regError = **false**;  
 }, 5000);  
 }  
  
 onSubmit() {  
 **if** (**this**.loginForm.valid) {  
 **const** {usernameOrEmail, password} = **this**.loginControls;  
 **this**.authenticationService.login(usernameOrEmail.value, password.value)  
 .pipe(first())  
 .subscribe(  
 (response) => {  
 **if** (response.principal.active) {  
 **this**.router.navigate(['scheme-pumpstation']);  
 } **else** { **this**.showError('Аккаунт не активирован!');  
 }  
 }, error => {  
 **this**.showError('Неправильные логин/email или пароль!');  
 });  
 } **else** {  
 **if** (**this**.loginForm.controls.usernameOrEmail.invalid &&  
 (**this**.loginForm.controls.usernameOrEmail || **this**.loginForm.controls.usernameOrEmail)) {  
 **if** (**this**.loginForm.controls.usernameOrEmail.errors.required) {  
 **this**.showError('Поле логин/email обязательное!');  
 }  
 **if** (**this**.loginForm.controls.usernameOrEmail.errors.minlength) {  
 **this**.showError('Логин должен быть не менее 5 символов');  
 }  
 **if** (**this**.loginForm.controls.usernameOrEmail.errors.pattern) {  
 **this**.showError('В поле логин используются запрещенные символы');  
 }  
 }  
 **if** (**this**.loginForm.controls.password.invalid && (**this**.loginForm.controls.password || **this**.loginForm.controls.password)) {  
 **if** (**this**.loginForm.controls.password.errors.required) {  
 **this**.showError('Поле пароль обязательное!');  
 }  
 **if** (**this**.loginForm.controls.password.errors.required && **this**.loginForm.controls.usernameOrEmail.errors.required) {  
 **this**.showError('Логин и пароль обязательны!');  
 }  
 **if** (**this**.loginForm.controls.password.errors.minlength) {  
 **this**.showError('Пароль должен быть не менее 8 символов');  
 }  
 **if** (**this**.loginForm.controls.password.errors.pattern) {  
 **this**.showError('В поле пароль используются запрещенные символы');  
 }  
 }  
 }  
 }  
  
 registration() {  
 **if** (**this**.registrationForm.valid) {  
 **this**.proccedRegistration = **true**;  
 **const** request = {  
 username: **this**.registrationControls.username.value,  
 password: **this**.registrationControls.password.value,  
 email: **this**.registrationControls.email.value  
 };  
 **this**.authenticationService.registrate(request).subscribe((apiResponse) => {  
 **if** (apiResponse.success) {  
 **this**.proccedRegistration = **false**;  
 **this**.modalOpened = **false**;  
 **this**.showSuccess('Вы зарегистрированы! Теперь необходимо активировать аккаунт.');  
 } **else** {  
 **this**.proccedRegistration = **false**;  
 **switch** (apiResponse.message) {  
 **case** 'Username is already taken!':  
 **this**.registrationError('Логин уже занят!');  
 **break**;  
 **case** 'Email Address already in use!':  
 **this**.registrationError('Email уже занят!');  
 **break**;  
 }  
 }  
 }, (error) => {  
 **this**.proccedRegistration = **false**;  
 **this**.modalOpened = **false**;  
 **this**.showError('Регистрация не удалась по непредвиденной ошибке. Возможно, не удалось отправить сообщение на указанную почту');  
 });  
 }  
 }  
}

login.component.spec.ts

**import** {async, ComponentFixture, TestBed} **from** '@angular/core/testing';  
  
**import** {LoginComponent} **from** './login.component';  
  
describe('LoginComponent', () => {  
 **let** component: LoginComponent;  
 **let** fixture: ComponentFixture<LoginComponent>;  
  
 beforeEach(async(() => {  
 TestBed.configureTestingModule({  
 declarations: [LoginComponent]  
 })  
 .compileComponents();  
 }));  
  
 beforeEach(() => {  
 fixture = TestBed.createComponent(LoginComponent);  
 component = fixture.componentInstance;  
 fixture.detectChanges();  
 });  
  
 it('should create', () => {  
 expect(component).toBeTruthy();  
 });  
});

authentication.service.ts

**import** {Injectable} **from** '@angular/core';  
**import** {HttpClient} **from** '@angular/common/http';  
**import** {BehaviorSubject, Observable} **from** 'rxjs';  
**import** {map} **from** 'rxjs/operators';  
**import** {AbstractControl} **from** '@angular/forms';  
**import** {AuthToken} **from** '../models/authToken';  
**import** {environment} **from** '../../../environments/environment';  
**import** {SignUpRequest} **from** '../models/sign.up.request';  
**import** {UserService} **from** './user.service';  
**import** {User} **from** '../models/user';  
  
@Injectable({providedIn: 'root'})  
**export class** AuthenticationService {  
 **private** currentAccessTokenSubject: BehaviorSubject<AuthToken>;  
 **public** currentAccessToken: Observable<AuthToken>;  
  
 **constructor**(**private** http: HttpClient) {  
 **const** authToken = JSON.parse(localStorage.getItem('authToken')) **as** AuthToken;  
 **this**.currentAccessTokenSubject = **new** BehaviorSubject<AuthToken>(authToken);  
 **this**.currentAccessToken = **this**.currentAccessTokenSubject.asObservable();  
 }  
  
 **public get** isLoggedIn() {  
 **return this**.currentAccessTokenValue.active && **this**.currentAccessTokenValue;  
 }  
  
 **public get** currentAccessTokenValue(): AuthToken {  
 **return this**.currentAccessTokenSubject.value;  
 }  
  
 login(usernameOrEmail: AbstractControl, password: AbstractControl) {  
 **return this**.http.post<**any**>(`${environment.apiUrl}/auth/signin`, {usernameOrEmail, password})  
 .pipe(map(response => {  
 **if** (response.principal.active) {  
 **const** authToken = {  
 accessToken: response.accessToken,  
 role: response.principal.role,  
 active: response.principal.active  
 } **as** AuthToken;  
 localStorage.setItem('authToken', JSON.stringify(authToken));  
 **this**.currentAccessTokenSubject.next(authToken);  
 }  
 **return** response;  
 }));  
 }  
  
 logout() {  
 localStorage.removeItem('authToken');  
 **this**.currentAccessTokenSubject.next(**null**);  
 }  
  
 registrate(signUpRequest: SignUpRequest) {  
 **return this**.http.post<**any**>(`${environment.apiUrl}/auth/signup`, signUpRequest);  
 }  
}

# 5. Компонент главного экрана станции

app.component.spec.ts

**import** {async, TestBed} **from** '@angular/core/testing';  
**import** {AppComponent} **from** './app.component';  
  
describe('AppComponent', () => {  
 beforeEach(async(() => {  
 TestBed.configureTestingModule({  
 declarations: [  
 AppComponent  
 ],  
 }).compileComponents();  
 }));  
  
 it('should create the app', () => {  
 **const** fixture = TestBed.createComponent(AppComponent);  
 **const** app = fixture.debugElement.componentInstance;  
 expect(app).toBeTruthy();  
 });  
  
 it(`should have as title 'asuns-front-app'`, () => {  
 **const** fixture = TestBed.createComponent(AppComponent);  
 **const** app = fixture.debugElement.componentInstance;  
 expect(app.title).toEqual('snsr-front-app');  
 });  
  
 it('should render title', () => {  
 **const** fixture = TestBed.createComponent(AppComponent);  
 fixture.detectChanges();  
 **const** compiled = fixture.debugElement.nativeElement;  
 expect(compiled.querySelector('.content span').textContent).toContain('asuns-front-app app is running!');  
 });  
});

app.module.ts

**import** {BrowserModule} **from** '@angular/platform-browser';  
**import** {NgModule} **from** '@angular/core';  
  
**import** {AppComponent} **from** './app.component';  
**import** {ClarityModule} **from** '@clr/angular';  
**import** {BrowserAnimationsModule} **from** '@angular/platform-browser/animations';  
**import** {RouterModule} **from** '@angular/router';  
**import** {RoutingModule} **from** './routing/routing.module';  
**import** {HTTP\_INTERCEPTORS, HttpClientModule} **from** '@angular/common/http';  
**import** {DatePipe} **from** '@angular/common';  
**import** {DefaultModule} **from** './layouts/default/default.module';  
**import** {PlotsComponent} **from** './modules/plots/plots.component';  
**import** {SharedModule} **from** './shared/shared.module';  
**import** {FormsModule, ReactiveFormsModule} **from** '@angular/forms';  
**import** {JwtInterceptor} **from** './shared/payload/jwt.interceptor';  
**import** {LoginComponent} **from** './modules/login/login.component';  
**import** {ManagementComponent} **from** './modules/management/management.component';  
**import** { SensorCreationComponent } **from** './modules/management/sensor-creation/sensor-creation.component';  
**import** { BaseStationCreationComponent } **from** './modules/management/base-station-creation/base-station-creation.component';  
**import** {ErrorInterceptor} **from** './shared/payload/error.interceptor';  
**import** {MatTableModule} **from** "@angular/material/table";  
  
@NgModule({  
 declarations: [  
 AppComponent,  
 PlotsComponent,  
 LoginComponent,  
 ManagementComponent,  
 SensorCreationComponent,  
 BaseStationCreationComponent,  
 ],  
 imports: [  
 BrowserModule,  
 ClarityModule,  
 BrowserAnimationsModule,  
 RouterModule,  
 RoutingModule,  
 HttpClientModule,  
 DefaultModule,  
 DefaultModule,  
 SharedModule,  
 ReactiveFormsModule,  
 FormsModule,  
 BrowserModule,  
 BrowserAnimationsModule,  
 HttpClientModule,  
 MatTableModule  
 ],  
 providers: [DatePipe,  
 {provide: HTTP\_INTERCEPTORS, useClass: JwtInterceptor, multi: **true**},  
 {provide: HTTP\_INTERCEPTORS, useClass: ErrorInterceptor, multi: **true**}  
 ],  
 bootstrap: [AppComponent]  
})  
**export class** AppModule {  
}

environment.ts

**export const** environment = {  
 apiUrl: 'http://localhost:8443/api',  
  
 production: **false**};

environment.prod.ts

**export const** environment = {  
 apiUrl: 'https://pm-asuns.ru/api',  
 production: **true**};