Universitatea Tehnică de Construcții București
Facultatea de Hidrotehnica

Proiect la PCLP III

Profesor Coordonator
Drd. Ing. Olteanu Gabriela
Ș.l. univ. dr. ing. Ramona – Oana FLANGEA

Student
Tudor Constantin-Mihai

Agenda personala

Universitatea Tehnică de Construcții București

Facultatea de Hidrotehnica

Profesor Coordonator
Drd. Ing. Olteanu Gabriela
Ș.l. univ. dr. ing. Ramona – Oana FLANGEA

Student
Tudor Constantin-Mihai

Cuprins

1. Introducere

1.1 Motivarea alegerii temei

Motivația principală a alegerii acestui proiect constă în echilibrul dintre simplitatea conceptuală a unei agende și complexitatea tehnică necesară pentru o implementare modernă. Tema oferă un cadru ideal pentru explorarea unor tehnologii de actualitate, precum Spring Boot și MongoDB, permițând modelarea unui flux de lucru complet: de la autentificarea securizată prin JWT, la gestionarea eficientă a conținutului HTML generat dinamic. Astfel, proiectul reprezintă o oportunitate de a demonstra integrarea unui ecosistem tehnologic robust într-o aplicație utilitară.

1.2 Obiective

Scopul principal al acestui proiect a fost dezvoltarea unei aplicații web funcționale pentru gestionarea notițelor personale, punând accent pe următoarele obiective:

- Autentificare sigură: Crearea unui sistem de login bazat pe JWT (JSON Web Tokens), care să asigure că fiecare utilizator își poate accesa doar propriile date într-un mod securizat.
- Gestionarea eficientă a datelor (CRUD): Implementarea funcțiilor de bază pentru orice aplicație de acest tip: crearea, vizualizarea, editarea și ștergerea notițelor (Create, Read, Update, Delete).
- Editor de text avansat: Integrarea unui editor de tip Rich Text (TipTap) care să permită formatarea textului în timp real, oferind o experiență de scriere mult mai plăcută decât un simplu câmp de text.
- Persistența datelor: Utilizarea unei baze de date NoSQL (MongoDB) pentru a stoca în siguranță documentele și paginile create de utilizator, asigurând o structură flexibilă a informațiilor.
- Arhitectură modernă: Separarea clară între partea de server (Spring Boot) și partea de client (React), facilitând comunicarea rapidă între cele două prin intermediul unui API REST.

2. Tehnologii folosite

2.1 Spring Boot

Spring boot este un framework Java folosit pentru dezvoltarea aplicațiilor backend, în special a API-urilor web. Acesta simplifică foarte mult configurarea aplicației, oferind setări automate și componente gata de utilizare. Permite rularea aplicației fără configurări complicate de server, deoarece include un server web integrat. Este ales frecvent în proiecte moderne datorită rapidității de dezvoltare și ușurinței în întreținere.

2.2 React

React este o bibliotecă JavaScript folosită pentru construirea interfețelor web interactive. Aceasta permite crearea aplicațiilor ca un ansamblu de componente reutilizabile, care se actualizează rapid atunci când datele se schimbă. React îmbunătățește experiența utilizatorului prin afișarea dinamică a conținutului, fără reîncărcarea paginii. Este foarte popular în aplicațiile moderne de tip single-page application.

2.3 MongoDB

MongoDB este o bază de date de tip NoSQL care stochează datele sub formă de documente JSON-like. Este flexibilă și permite lucrul cu date nestructurate sau care se pot schimba ușor în timp. MongoDB este des utilizată în aplicații web moderne datorită performanței bune și integrării ușoare cu aplicațiile backend. Este potrivită pentru proiecte care necesită scalabilitate și rapiditate.

2.4 Brave Browser

Brave Browser este un browser modern care pune accent pe confidențialitate, blocând implicit reclamele și trackerele. Din punctul de vedere al web developmentului, Brave se comportă foarte asemănător cu Google Chrome, deoarece este bazat pe Chromium. Aplicațiile web

funcționează în general fără modificări, însă anumite scripturi de tracking, reclame sau cookie-uri pot fi blocate automat. De aceea, este util pentru testarea aplicațiilor în condiții de confidențialitate ridicată.

2.5 IntelliJ IDEA

IntelliJ IDEA este un mediu de dezvoltare integrat (IDE) folosit în special pentru programarea în Java. Acesta oferă funcții avansate precum completarea automată a codului, detectarea erorilor și instrumente de debugging. Din punctul de vedere al dezvoltării web, este foarte util pentru proiecte backend, în special cu Spring Boot. IntelliJ ajută la scrierea unui cod mai curat și mai eficient, crescând productivitatea dezvoltatorului.

2.6 Visual Studio Code

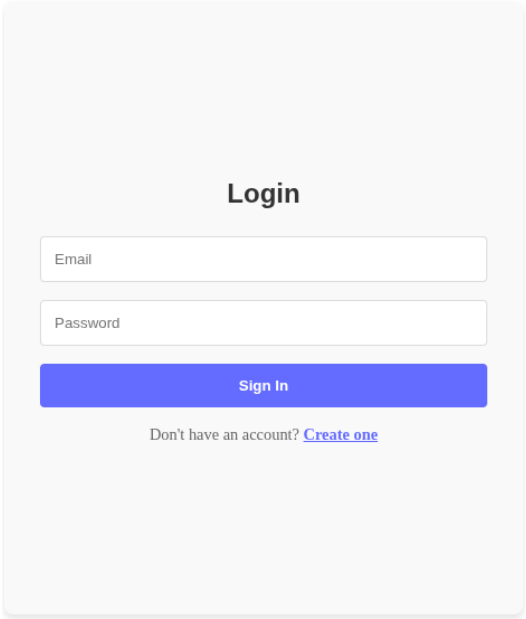
Visual Studio Code este un editor de cod ușor și flexibil, foarte popular în dezvoltarea web. Acesta suportă numeroase limbaje de prfogramare și poate fi extins prin extensii pentru frontend și backend. VS Code oferă funcții precum evidențierea sintaxei, completare automată și debugging. Este apreciat pentru rapiditate și pentru ușurința cu care poate fi adaptat la diferite tipuri de proiecte.

2.7 GitHub

GitHub este o platformă online folosită pentru gestionarea și stocarea proiectelor de software. Aceasta permite colaborarea între dezvoltatori prin folosirea sistemului de control al versiunilor Git. GitHub ajută la urmărirea modificărilor din cod, rezolvarea conflictelor și organizarea muncii în echipă. Este esențial în dezvoltarea modernă a aplicațiilor software.

3. Scenariu de functionare

3.1 Interfața de Autentificare și Securitatea Datelor



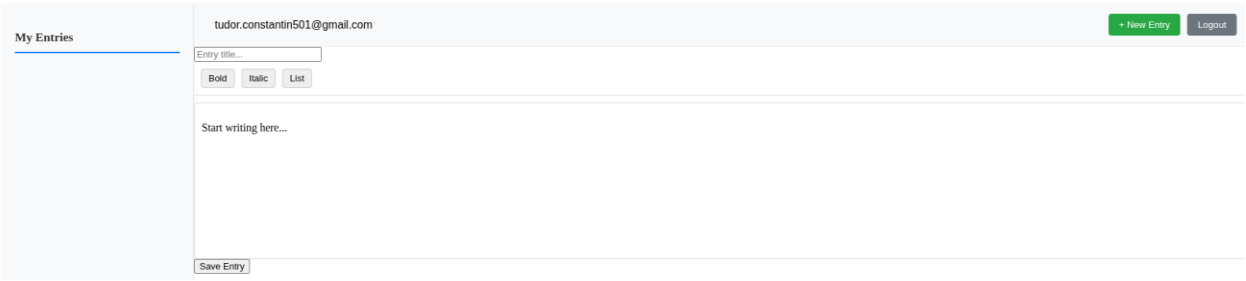
Interfața de login a fost concepută să fie cât mai intuitivă și curată, eliminând elementele inutile pentru a nu distrage utilizatorul. Aceasta oferă o experiență de utilizare simplă: utilizatorul introduce adresa de e-mail și parola, iar aplicația procesează datele instantaneu.

În spatele acestei interfețe simple, am implementat un mecanism de securitate modern bazat pe tokene JWT (JSON Web Tokens). Procesul funcționează astfel:

- Identificare sigură: În momentul logării, serverul verifică datele și trimite înapoi un token unic (o cheie digitală).
- Acces persistent: Acest token este salvat în browser, permițând utilizatorului să rămână conectat chiar dacă dă refresh paginii, fără a fi nevoie să introducă parola de fiecare dată.
- Protecția datelor: Toate cererile către baza de date (pentru a vedea sau salva notițe) sunt verificate prin acest token. Dacă cineva încearcă să acceseze datele fără o cheie validă, serverul va bloca automat accesul.

- Feedback vizual: Interfața este gândită să informeze utilizatorul în timp real (prin mesaje de eroare sau succes) dacă datele introduse sunt corecte sau dacă a apărut o problemă la conectare.

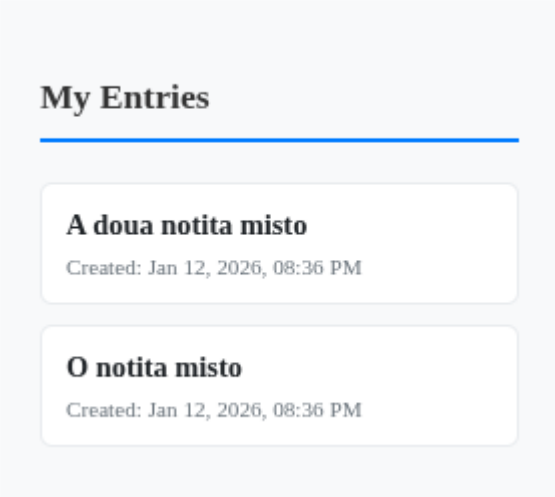
3.2 Interfața Principală și Spațiul de Lucru



După autentificare, utilizatorul este direcționat către spațiul de lucru principal, care a fost proiectat pentru productivitate și claritate. Interfața este împărțită strategic în două secțiuni:

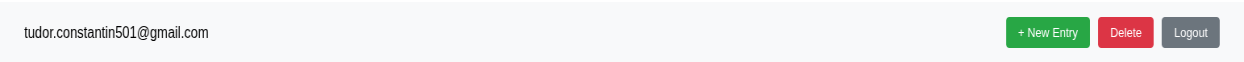
- Widget-ul Sidebar: În partea stângă se află o listă cu toate notițele salvate, organizate cronologic. Acest widget funcționează ca un meniu rapid: printr-un simplu click pe un titlu, aplicația aduce instant conținutul din baza de date MongoDB direct în editor.
- Zona de Editare: Partea centrală este ocupată de editorul propriu-zis. Spre deosebire de un simplu câmp de text, acesta permite formatări precum Bold, *Italic* sau liste, salvând tot conținutul sub formă de structură HTML în backend.
- Bara Superioară: Aceasta afișează utilizatorul conectat și oferă acces rapid la funcțiile de administrare, precum butonul de "+ New Entry" pentru a începe o notiță nouă sau butonul de Logout pentru încheierea sesiunii securizate.
- Sincronizarea Datelor: Interfața comunică permanent cu serverul prin cereri asincrone. De exemplu, când apeși pe o notiță în sidebar, în spate se execută o cerere către endpoint-ul de backend care folosește EntryResponseDTO pentru a încărca textul corect, fără a reîncărca întreaga pagină.

3.3 Widget-ul Sidebar



Widget-ul Sidebar reprezintă componenta centrală de navigare a agendei, fiind conceput pentru a oferi un acces rapid și intuitiv la istoricul personal de notițe. Acesta afișează în mod organizat titlurile și datele de creare pentru fiecare înregistrare, permițând utilizatorului să găsească imediat informația căutată. Printr-o simplă selecție în listă, widget-ul comunică eficient cu serverul pentru a încărca automat conținutul în zona de editare, fără a reîncărca pagina. Designul său minimalist și curat transformă agenda într-un spațiu de lucru bine structurat, unde gestionarea datelor salvate în baza de date devine o acțiune simplă și rapidă.

3.4 Header-ul Aplicației



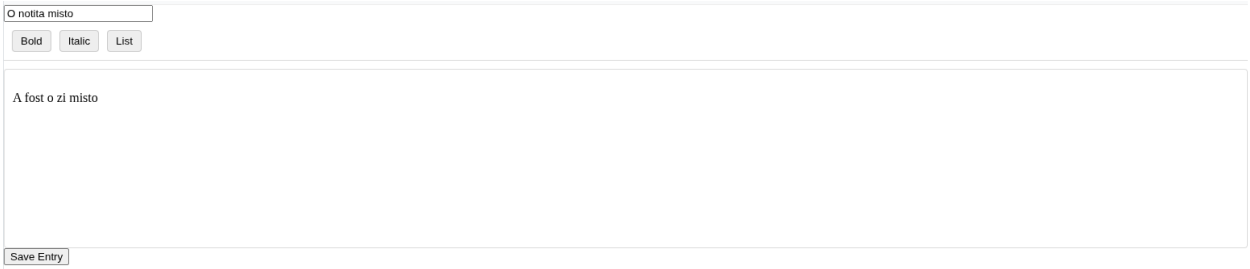
Header-ul este zona superioară a interfeței care servește drept punct central pentru gestionarea sesiunii și a acțiunilor globale. Acesta este conceput să fie minimalist, oferind utilizatorului acces imediat la funcțiile esențiale fără a încărca vizual spațiul de lucru.

Principalele elemente integrate în header sunt:

- Identificarea Utilizatorului: În partea stângă este afișată adresa de e-mail a utilizatorului conectat, confirmând identitatea sesiunii curente securizate prin JWT.

- Acțiuni de Gestiune: În partea dreaptă sunt grupate butoanele de acțiune rapidă, precum "+ New Entry" pentru crearea unei notițe noi și butonul de "Delete" pentru eliminarea rapidă a înregistrărilor selectate.
- Controlul Sesiunii: Butonul de "Logout" permite închiderea securizată a aplicației, asigurând deconectarea utilizatorului și ștergerea token-ului de acces din browser.
- Design și Integrare: Bara superioară menține o estetică modernă și aerisită, fiind separată vizual de restul conținutului pentru a delimita clar zona de administrare de cea de editare propriu-zisă.

3.5 Editorul de Text



Editorul de text reprezintă componenta centrală a spațiului de lucru, fiind locul unde utilizatorul își poate redacta și formata ideile într-un mod flexibil. Spre deosebire de un câmp de text obișnuit, am integrat editorul TipTap, care permite transformarea conținutului într-o structură Rich Text ce suportă stilizări precum îngroșarea textului, scrierea cursivă sau crearea de liste. Această funcționalitate asigură o experiență de scriere dinamică, utilizatorul având la dispoziție o bară de unelte intuitivă pentru a-și organiza vizual notițele. Toate modificările efectuate sunt procesate în timp real și trimise către backend sub formă de conținut HTML, care este apoi salvat în baza de date MongoDB. Prin acest editor, aplicația depășește bariera unei agende clasice, oferind un spațiu de creație modern unde informația este stocată fidel formei în care a fost redactată.

4. Arhitectura Sistemului și Implementare

4.1 Entry.java

```java
package costijk.PersonalAgenda.model;

import ...

@Data  16 usages  Costijk
@Document(collection = "entries")
@CompoundIndex(name = "user_creation_idx", def = "{'userId' : 1, 'createdAt' : -1}")
public class Entry {

    @Id
    private String id;

    private String userId;

    private String title;

    private LocalDateTime createdAt = LocalDateTime.now();

    private LocalDateTime updatedAt;


    private List<Page> pages;
}
```

Clasa reprezintă o notiță a unui utilizator care se salvează în MongoDB; fiecare obiect conține un identificator, id-ul proprietarului, un titlu, o listă de pagini cu conținutul notei și timestamp-uri pentru creare și actualizare (createdAt e setat la crearea obiectului, updatedAt se folosește pentru modificări ulterioare). Există un index compus pentru a permite interogări rapide ale notițelor unui utilizator ordonate după cele mai noi, iar Lombok (@Data) elimină boilerplate-ul pentru gettere/settere și metode comune.

4.2 EntryRepository.java

```java
package costijk.PersonalAgenda.repository;

import ...

public interface EntryRepository extends MongoRepository<Entry, String> {  2 usages  Costijk
    Page<Entry> findByUserId(String userId, Pageable pageable);  1 usage  Costijk
}
```

EntryRepository este componenta care face legătura directă între codul aplicației și baza de date MongoDB. Aceasta extinde o interfață standard numită MongoRepository, care îi oferă automat toate funcțiile de bază pentru gestionarea datelor, cum ar fi salvarea, ștergerea sau căutarea notițelor.

În plus, am definit o metodă specială numită findByUserId, care are rolul de a filtra notițele astfel încât fiecare utilizator să își vadă doar propriile însemnări. Această metodă folosește și un sistem de paginare, ceea ce înseamnă că lista de notițe din widget este încărcată eficient, în porțiuni mici, fără a suprasolicita serverul atunci când agenda devine foarte încărcată.

4.3 EntryService.java

```java
package costijk.PersonalAgenda.service;

import ...

@Service  2 usages  Costijk
@RequiredArgsConstructor
public class EntryService {

    private final EntryRepository entryRepository;
    private final UserRepository userRepository;


    private String getCurrentUserId() {  4 usages  Costijk
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        if (authentication == null || !authentication.isAuthenticated()) {
            throw new RuntimeException("User not authenticated!");
        }

        Object principal = authentication.getPrincipal();

        if (principal instanceof costijk.PersonalAgenda.model.User) {
            return ((costijk.PersonalAgenda.model.User) principal).getId();
        }

        String email = authentication.getName();
        return userRepository.findByEmail(email) Optional<User>
                .orElseThrow(() -> new RuntimeException("User not found")) User
                .getId();
    }


    public EntryResponseDTO createEntry(EntryCreationDTO creationDTO) {  1 usage  Costijk
        String userId = getCurrentUserId();
        Entry entry = convertToEntity(creationDTO, userId);
        Entry savedEntry = entryRepository.save(entry);
        return convertToResponseDTO(savedEntry);
    }

    public org.springframework.data.domain.Page<NoteListDTO> getMyEntriesPaginated(int page, int size) {  1 usage  Costijk
        Pageable pageable = PageRequest.of(page, size, Sort.by( ...properties: "createdAt").descending());
```

```java
public class EntryService {

    public org.springframework.data.domain.Page<NoteListDTO> getMyEntriesPaginated(int page, int size) {  1 usage  Costijk
        Pageable pageable = PageRequest.of(page, size, Sort.by( ...properties: "createdAt").descending());

        org.springframework.data.domain.Page<Entry> entriesPage = entryRepository.findByUserId(getCurrentUserId(), pageable);

        return entriesPage.map( Entry entry -> {
            NoteListDTO dto = new NoteListDTO();
            dto.setId(entry.getId());
            dto.setTitle(entry.getTitle());
            dto.setCreatedAt(entry.getCreatedAt());
            dto.setUpdatedAt(entry.getUpdatedAt());
            return dto;
        });
    }

@   private Entry convertToEntity(EntryCreationDTO creationDTO, String currentUserId) {  1 usage  Costijk
        Entry entry = new Entry();
        entry.setUserId(currentUserId);
        entry.setTitle(creationDTO.getTitle());
        entry.setCreatedAt(LocalDateTime.now());

        if (creationDTO.getPages() != null) {
            entry.setPages(creationDTO.getPages().stream().map( PageDTO pageDto -> {
                costijk.PersonalAgenda.model.Page page = new costijk.PersonalAgenda.model.Page();
                page.setPageNumber(pageDto.getPageNumber());
                page.setContent(pageDto.getContent());
                return page;
            }).collect(Collectors.toList()));
        }
        return entry;
    }

@   private EntryResponseDTO mapToResponseDTO(Entry entry) {  no usages  Costijk
        EntryResponseDTO dto = new EntryResponseDTO();
        dto.setId(entry.getId());
        dto.setTitle(entry.getTitle());

        if (entry.getPages() != null && !entry.getPages().isEmpty()) {
            dto.setContent(entry.getPages().get(0).getContent());
```

```java
public class EntryService {
    public EntryResponseDTO getEntryById(String id) {  1 usage  Costijk
        return convertToResponseDTO(entry);
    }

    public EntryResponseDTO updateEntry(String id, EntryCreationDTO updateDTO) {  1 usage  Costijk
        Entry entry = entryRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Entry not found"));

        String currentUserId = getCurrentUserId();
        if (!entry.getUserId().equals(currentUserId)) {
            throw new RuntimeException("You don't have permission to modify this entry");
        }

        entry.setTitle(updateDTO.getTitle());
        entry.setUpdatedAt(LocalDateTime.now());

        if (updateDTO.getPages() != null) {
            entry.setPages(updateDTO.getPages().stream().map( PageDTO pageDto -> {
                costijk.PersonalAgenda.model.Page page = new costijk.PersonalAgenda.model.Page();
                page.setPageNumber(pageDto.getPageNumber());
                page.setContent(pageDto.getContent());
                return page;
            }).collect(Collectors.toList()));
        }

        Entry updatedEntry = entryRepository.save(entry);
        return convertToResponseDTO(updatedEntry);
    }

    public void deleteEntry(String id) {  1 usage  Costijk
        Entry entry = entryRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Entry not found"));

        String currentUserId = getCurrentUserId();
        if (!entry.getUserId().equals(currentUserId)) {
            throw new RuntimeException("You don't have permission to delete this entry");
        }

        entryRepository.delete(entry);
    }
```

EntryService funcționează ca motorul logic al aplicației, fiind responsabil pentru procesarea tuturor operațiunilor ce țin de gestionarea notițelor. Această clasă asigură securitatea datelor prin verificarea identității utilizatorului curent la fiecare acțiune, garantând astfel că nicio notiță nu poate fi accesată, modificată sau ștearsă de către o altă persoană. Pe lângă gestionarea funcțiilor de bază, serviciul se ocupă de organizarea eficientă a informațiilor prin sortarea și paginarea rezultatelor trimise către interfață, asigurând o încărcare rapidă a listei de notițe. De asemenea, joacă un rol esențial în transformarea datelor, convertind structura complexă a paginilor din baza de date într-un format simplificat și ușor de procesat de către editorul din frontend.

4.4 EntryController.java

```java
package costijk.PersonalAgenda.controller;

import ...

@RestController  Costijk
@RequestMapping("/api/entries")
@RequiredArgsConstructor
public class EntryController {

    private final EntryService entryService;

    @PostMapping  Costijk
    public ResponseEntity<EntryResponseDTO> createEntry(@RequestBody EntryCreationDTO dto) {
        return new ResponseEntity<>(entryService.createEntry(dto), HttpStatus.CREATED);
    }

    @GetMapping("/list")  Costijk
    public ResponseEntity<Page<NoteListDTO>> getEntriesList(
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "10") int size) {

        return ResponseEntity.ok(entryService.getMyEntriesPaginated(page, size));
    }

    @GetMapping("/{id}")  Costijk
    public ResponseEntity<EntryResponseDTO> getEntryById(@PathVariable String id) {

        return ResponseEntity.ok(entryService.getEntryById(id));
    }

    @PutMapping("/{id}")  Costijk
    public ResponseEntity<EntryResponseDTO> updateEntry(@PathVariable String id, @RequestBody EntryCreationDTO dto) {
        return ResponseEntity.ok(entryService.updateEntry(id, dto));
    }

    @DeleteMapping("/{id}")  Costijk
    public ResponseEntity<Void> deleteEntry(@PathVariable String id) {
        entryService.deleteEntry(id);
        return ResponseEntity.noContent().build();
    }
}
```

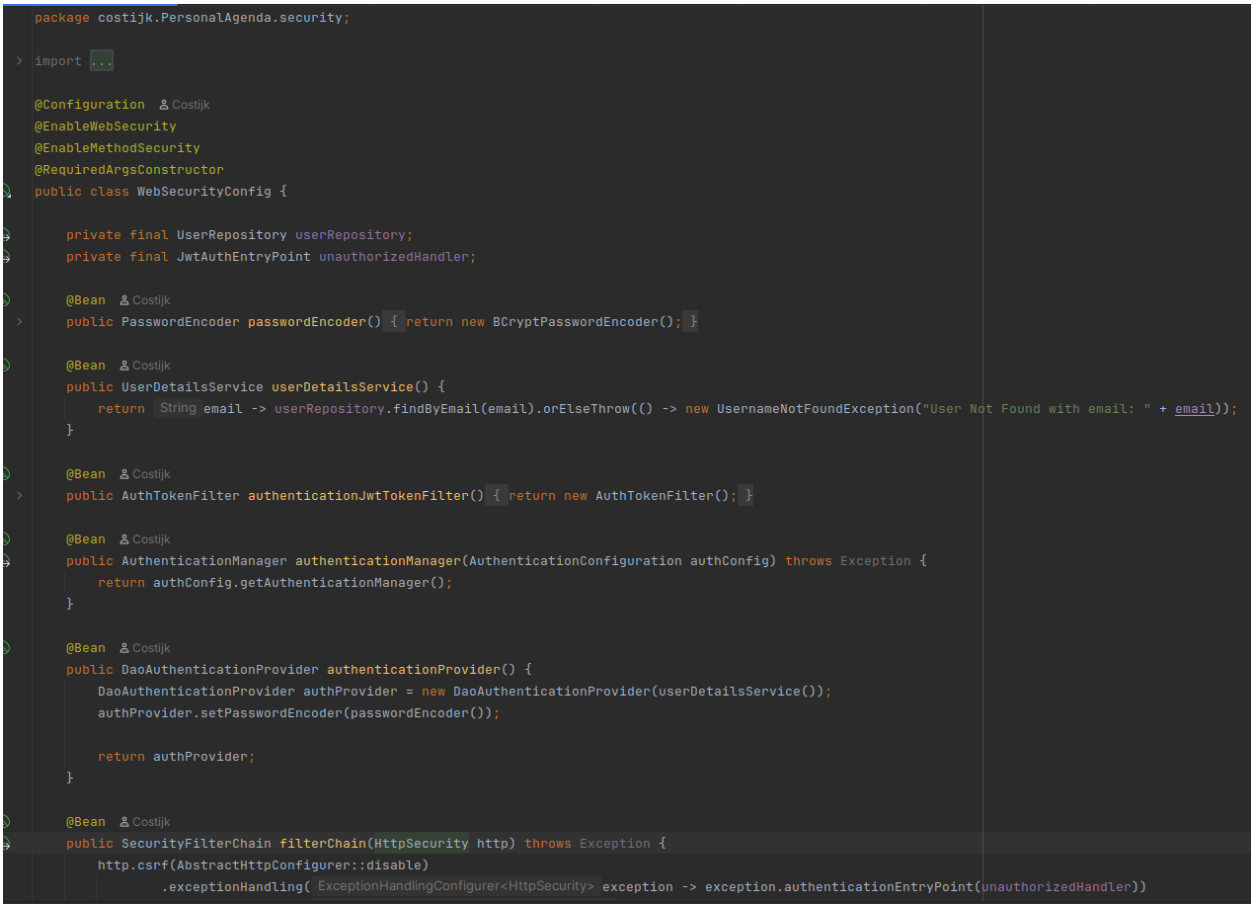
EntryController acționează ca punctul principal de contact între interfața utilizatorului (Frontend) și logica internă a aplicației (Backend), fiind responsabil pentru gestionarea tuturor cererilor de tip API. Această clasă definește rutele specifice prin care aplicația primește instrucțiuni, cum ar fi crearea unei notițe noi, descărcarea listei de însemnări sau ștergerea unui document. Controlerul preia datele trimise de utilizator, le direcționează către serviciul corespunzător pentru procesare și apoi returnează un răspuns clar, confirmând succesul operațiunii sau trimițând înapoi informația solicitată. Prin utilizarea metodelor standard de comunicare web, acesta permite aplicației React să interacționeze rapid și sigur cu serverul, asigurând o experiență de utilizare fluidă și sincronizată.

4.5 EntryResponseDTO.java

```java
package costijk.PersonalAgenda.dto;

import lombok.Data;
import java.time.LocalDateTime;
import java.util.List;

@Data  14 usages  Costijk
public class EntryResponseDTO {
    private String id;
    private String title;
    private LocalDateTime createdAt;
    private LocalDateTime updatedAt;
    private String content;
    private List<PageDTO> pages;
}
```

EntryResponseDTO este o clasă special creată pentru a filtra și organiza informațiile care pleacă de la server către interfața utilizatorului. Rolul acestui obiect este de a asigura un transfer de date eficient, trimițând către frontend doar câmpurile necesare, precum titlul, conținutul și datele de timp, fără a expune detalii tehnice interne ale bazei de date. Prin gruparea paginilor și a conținutului într-un format simplu, această clasă permite aplicației React să afișeze instantaneu informațiile în editor și în lista de navigare, menținând o structură clară și ușor de procesat. Practic, acest obiect funcționează ca un mesager care traduce datele complexe din backend într-un format pe care interfața îl poate desena imediat pe ecran.

4.6 WebSecurityConfig.java



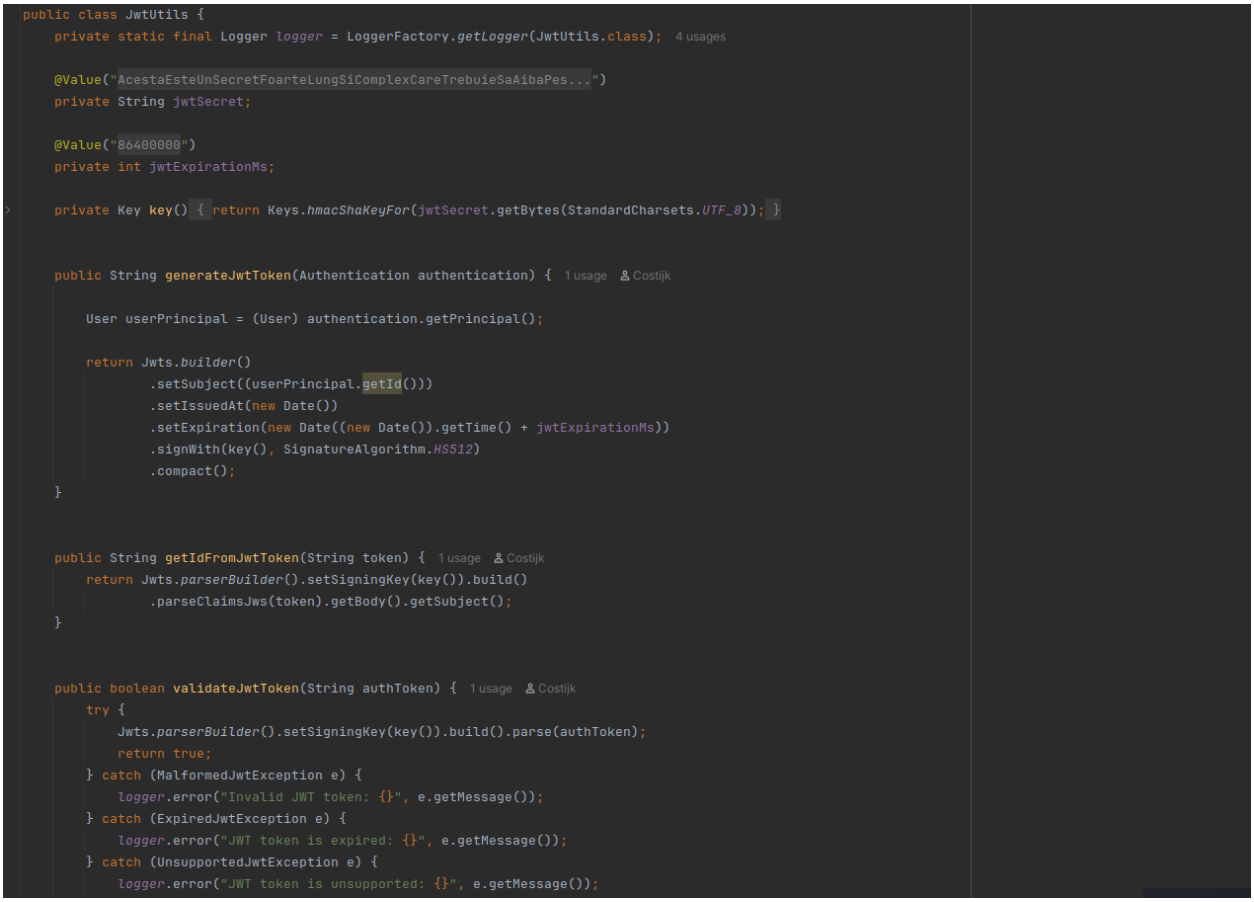
```java
package costijk.PersonalAgenda.security;

import ...

@Configuration  Costijk
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class WebSecurityConfig {

    private final UserRepository userRepository;
    private final JwtAuthEntryPoint unauthorizedHandler;

    @Bean  Costijk
    public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    @Bean  Costijk
    public UserDetailsService userDetailsService() {
        return String email -> userRepository.findByEmail(email).orElseThrow(() -> new UsernameNotFoundException("User Not Found with email: " + email));
    }

    @Bean  Costijk
    public AuthTokenFilter authenticationJwtTokenFilter() { return new AuthTokenFilter(); }

    @Bean  Costijk
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authConfig) throws Exception {
        return authConfig.getAuthenticationManager();
    }

    @Bean  Costijk
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider(userDetailsService());
        authProvider.setPasswordEncoder(passwordEncoder());

        return authProvider;
    }

    @Bean  Costijk
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
                .exceptionHandling( ExceptionHandlingConfigurer<HttpSecurity> exception -> exception.authenticationEntryPoint(unauthorizedHandler))
```

```java
    @Bean  Costijk
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf(AbstractHttpConfigurer::disable)
                .exceptionHandling( ExceptionHandlingConfigurer<HttpSecurity> exception -> exception.authenticationEntryPoint(unauthorizedHandler))
                .sessionManagement( SessionManagementConfigurer<HttpSecurity> session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests( AuthorizationManagerRequestMat... auth -> auth
                        .requestMatchers("/api/auth/**").permitAll()
                        .requestMatchers("/error").permitAll()
                        .anyRequest().authenticated()
                );

        http.authenticationProvider(authenticationProvider());
        http.addFilterBefore(authenticationJwtTokenFilter(), UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}
```

WebSecurityConfig este clasa care definește regulile de protecție ale aplicației, funcționând ca un sistem de pază care decide cine are voie să acceseze resursele serverului. Aceasta configurează securitatea astfel încât paginile de login și înregistrare să fie accesibile tuturor, în timp ce restul funcțiilor, precum

gestionarea notițelor, necesită o identitate confirmată. Clasa se ocupă de verificarea parolelor folosind un algoritm de criptare sigur și integrează sistemul de tokene JWT, care permite utilizatorului să rămână conectat fără a trimite parola la fiecare pas. De asemenea, această componentă elimină necesitatea sesiunilor clasice pe server, făcând aplicația mai rapidă și mai modernă prin verificarea automată a fiecărei cereri care vine de la interfață. În esență, este locul unde am stabilit că nicio informație privată nu poate fi citită sau modificată fără o "cheie digitală" validă.

4.7 JwtUtils.java



```
public class JwtUtils {
    private static final Logger logger = LoggerFactory.getLogger(JwtUtils.class);  4 usages

    @Value("AcestaEsteUnSecretFoarteLungSiComplexCareTrebuieSaAibaPes...")
    private String jwtSecret;

    @Value("86400000")
    private int jwtExpirationMs;

    private Key key() { return Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardCharsets.UTF_8)); }


    public String generateJwtToken(Authentication authentication) {  1 usage  Costijk

        User userPrincipal = (User) authentication.getPrincipal();

        return Jwts.builder()
                .setSubject((userPrincipal.getId()))
                .setIssuedAt(new Date())
                .setExpiration(new Date((new Date()).getTime() + jwtExpirationMs))
                .signWith(key(), SignatureAlgorithm.HS512)
                .compact();
    }


    public String getIdFromJwtToken(String token) {  1 usage  Costijk
        return Jwts.parserBuilder().setSigningKey(key()).build()
                .parseClaimsJws(token).getBody().getSubject();
    }


    public boolean validateJwtToken(String authToken) {  1 usage  Costijk
        try {
            Jwts.parserBuilder().setSigningKey(key()).build().parse(authToken);
            return true;
        } catch (MalformedJwtException e) {
            logger.error("Invalid JWT token: {}", e.getMessage());
        } catch (ExpiredJwtException e) {
            logger.error("JWT token is expired: {}", e.getMessage());
        } catch (UnsupportedJwtException e) {
            logger.error("JWT token is unsupported: {}", e.getMessage());
```

JwtUtils este componenta responsabilă cu gestionarea efectivă a „cheilor digitale” (tokenele JWT) care permit accesul securizat în aplicație. Această clasă are rolul de a genera un token unic în momentul în care un utilizator se loghează cu succes, incluzând în acesta o dată de expirare pentru a preveni utilizarea lui neautorizată pe termen lung. Pe lângă crearea tokenelor, clasa conține și logica necesară pentru a valida fiecare cerere care vine de la frontend, verificând dacă semnătura digitală este autentică și dacă accesul este încă permis. În cazul în care un token este expirat sau modificat, JwtUtils identifică

eroarea și blochează accesul la datele private, asigurând astfel că identitatea utilizatorului este protejată în permanență.

5. Bibliografie

https://www.mongodb.com/

https://search.brave.com/search?q=intelij

https://code.visualstudio.com/

https://brave.com/

https://react.dev/

https://spring.io/projects/spring-boot

https://github.com/

https://www.youtube.com/watch?v=KuM6OtuaYRs&list=PLGRDMO4rOGcNLnW1L2vgsExTBg-VPoZHr

https://www.youtube.com/watch?v=5PdEmeopJVQ&t=3012s

https://www.youtube.com/watch?v=gJrjgg1KVL4&t=1737s

https://www.geeksforgeeks.org/advance-java/spring-boot/

https://www.youtube.com/watch?v=MHn66JJH5zs&list=PLSsAz5wf2lkK_ekd0J__44KG6QoXetZza