

Emergency and Mental Wellbeing

Project Statement:

The Emergency and Wellness Management System addresses the growing need for personal safety and mental well-being. This platform allows users to report emergencies, receive notifications, and track their mental health through assessments. It also provides personalized recommendations and access to support resources for improved well-being and safety.

Milestone – 1:(User Authentication)

- Implement user registration functionality.
- Develop user login mechanism.

Dependencies:

→Lombok

→spring data JPA

→Spring security

→Spring web

→MySQL driver

→Oauth

Packages & Classes:

→Controller

- AuthController

→DTO

- LoginDto
- RegisterDto
- UpdateDto

→Config

- SecurityConfig

→Model

- Person

→Repository

- PersonRepository

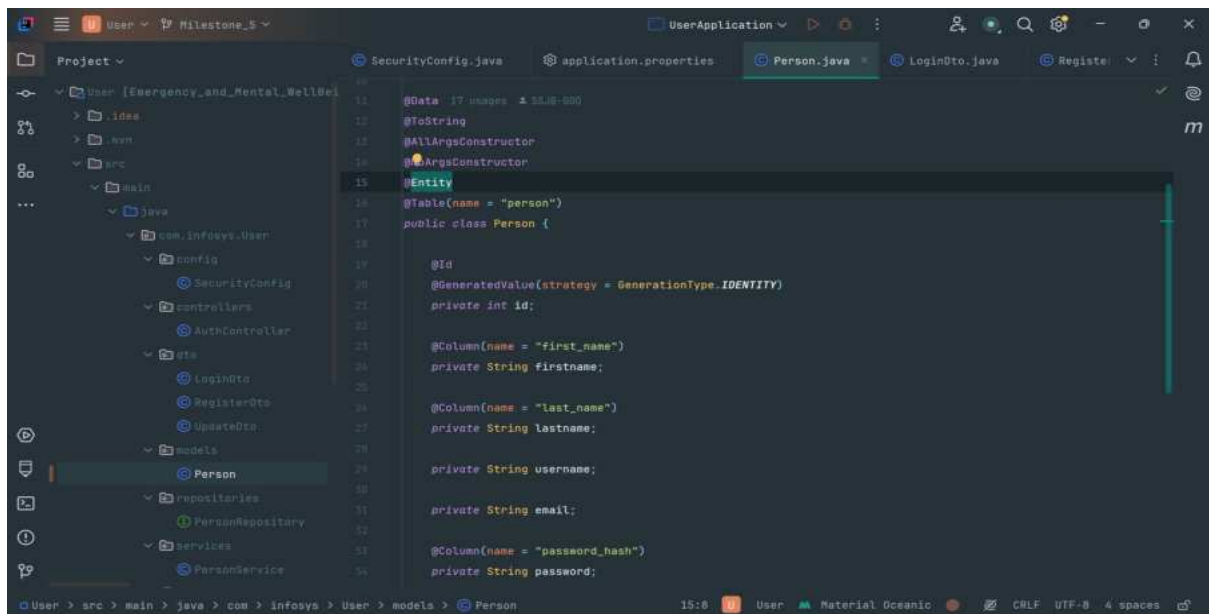
→Service

- PersonService

Model :

Define models to represent user data.

Class: Person

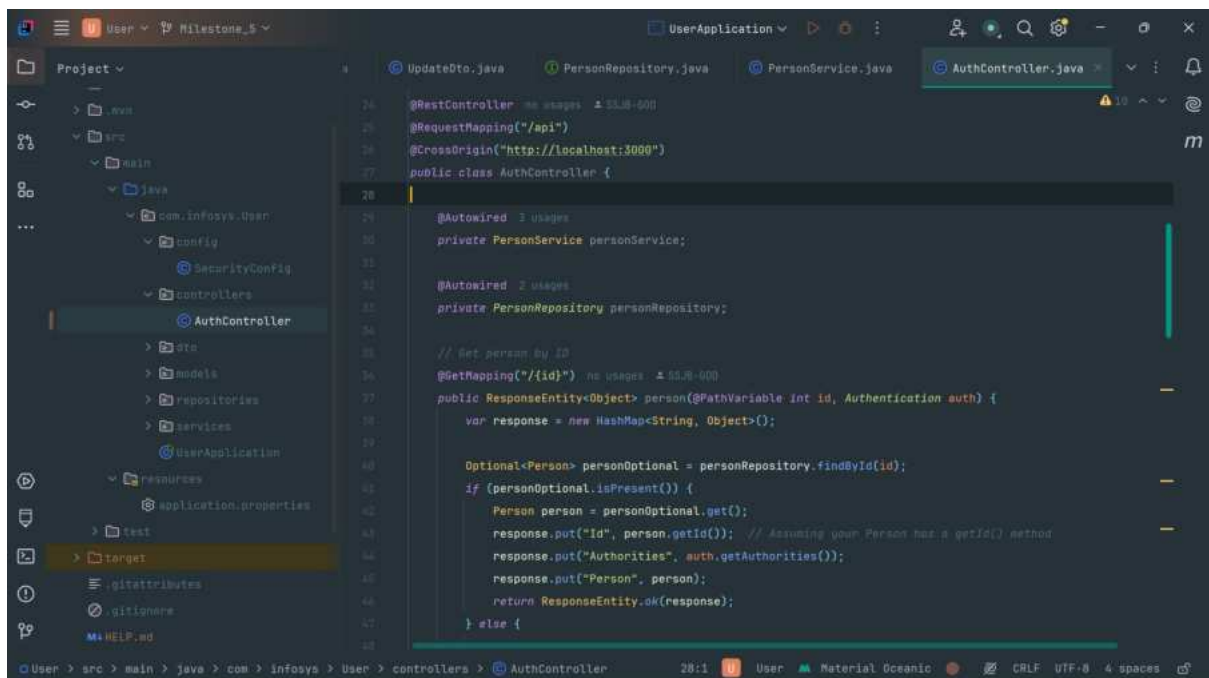


```
11 @Data
12 @ToString
13 @AllArgsConstructor
14 @NoArgsConstructor
15 @Entity
16 @Table(name = "person")
17 public class Person {
18
19     @Id
20     @GeneratedValue(strategy = GenerationType.IDENTITY)
21     private int id;
22
23     @Column(name = "first_name")
24     private String first_name;
25
26     @Column(name = "last_name")
27     private String last_name;
28
29     private String username;
30
31     private String email;
32
33     @Column(name = "password_hash")
34     private String password;
```

Controller:

Define Controller to HTTP requests.

Class: AuthController



```
24 @RestController
25 @RequestMapping("/api")
26 @CrossOrigin("http://localhost:3000")
27 public class AuthController {
28
29     @Autowired
30     private PersonService personService;
31
32     @Autowired
33     private PersonRepository personRepository;
34
35     // Get person by ID
36     @GetMapping("/{id}")
37     public ResponseEntity<Object> person(@PathVariable int id, Authentication auth) {
38         var response = new HashMap<String, Object>();
39
40         Optional<Person> personOptional = personRepository.findById(id);
41         if (personOptional.isPresent()) {
42             Person person = personOptional.get();
43             response.put("Id", person.getId()); // Assuming your Person has a getId() method
44             response.put("Authorities", auth.getAuthorities());
45             response.put("Person", person);
46             return ResponseEntity.ok(response);
47         } else {
48
49         }
```

This screenshot shows the 'register' method in the 'AuthController' class. The method is annotated with '@PostMapping("/register")' and takes a 'Valid @RequestBody RegisterDto registerDto' and a 'BindingResult result' as parameters. It first checks for validation errors using 'result.hasErrors()'. If there are errors, it creates an 'errorMap' by streaming all errors and collecting them into a map, where each key is the error's field and the value is its default message. It then returns a 'ResponseEntity.badRequest()' with this error map. If there are no errors, it calls 'personService.registerUser(registerDto)'. If the response contains the key 'error', it returns a bad request; otherwise, it returns an OK response with the response object.

```
public class AuthController {  
  
    @PostMapping("/register") no usages ▲ 55:18-60:0  
    public ResponseEntity<Object> register(@Valid @RequestBody RegisterDto registerDto, BindingResult result) {  
  
        if (result.hasErrors()) {  
            var errorMap = result.getAllErrors().stream()  
                .collect(Collectors.toMap(  
                    error -> ((FieldError) error).getField(),  
                    ObjectError::getDefaultMessage  
                ));  
            return ResponseEntity.badRequest().body(errorMap);  
        }  
  
        Map<String, Object> response = personService.registerUser(registerDto);  
        if (response.containsKey("error")) {  
            return ResponseEntity.badRequest().body(response);  
        }  
        return ResponseEntity.ok(response);  
    }  
  
    @PostMapping("/login") no usages ▲ 55:18-60:0  
    public ResponseEntity<Object> login(@Valid @RequestBody LoginDto loginDto, BindingResult result) {  
  
    }  
  
    @PostMapping("/update/{username}") no usages ▲ 55:18-60:0  
}
```

This screenshot shows the 'login' method in the 'AuthController' class. It is annotated with '@PostMapping("/login")' and takes a 'Valid @RequestBody LoginDto loginDto' and a 'BindingResult result' as parameters. It follows a similar validation pattern to the 'register' method. If there are validation errors, it returns a bad request with an error map. If not, it calls 'personService.authenticateUser(loginDto)'. If the response contains the key 'error', it prints 'Hello!' to the console and returns a bad request. Otherwise, it finds the user by email using 'personRepository.findById(loginDto.getEmail())', adds the user's ID to the response, and returns an OK response.

```
public class AuthController {  
  
    @PostMapping("/login") no usages ▲ 55:18-60:0  
    public ResponseEntity<Object> login(@Valid @RequestBody LoginDto loginDto, BindingResult result) {  
  
        if (result.hasErrors()) {  
            var errorMap = result.getAllErrors().stream()  
                .collect(Collectors.toMap(  
                    error -> ((FieldError) error).getField(),  
                    ObjectError::getDefaultMessage  
                ));  
            return ResponseEntity.badRequest().body(errorMap);  
        }  
  
        Map<String, Object> response = new HashMap<>();  
        if (personService.authenticateUser(loginDto).containsKey("error")) {  
            System.out.println("Hello!");  
            return ResponseEntity.badRequest().body(response);  
        }  
  
        Optional<Person> person = personRepository.findById(loginDto.getEmail());  
        response.put("id", person.get().getId());  
        return ResponseEntity.ok(response);  
    }  
  
    @PostMapping("/update/{username}") no usages ▲ 55:18-60:0  
}
```

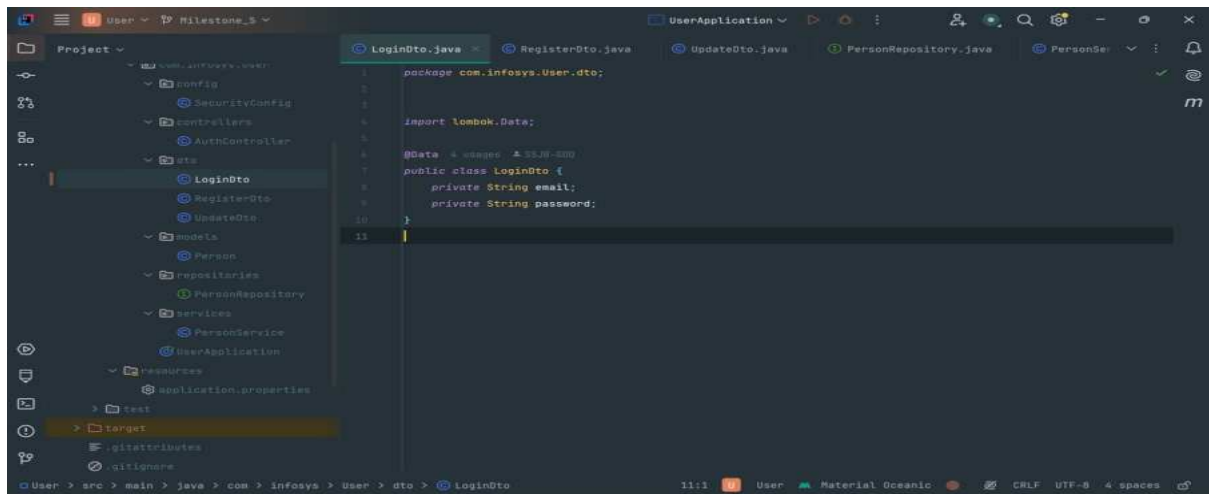
This screenshot shows the 'updateUser' method in the 'AuthController' class. It is annotated with '@PostMapping("/update/{username}")' and takes a 'PathVariable String username' and a 'RequestBody UpdateDto updateDto' as parameters. It calls 'personService.updateUser(username, updateDto)'. If the response contains the key 'error', it returns a bad request; otherwise, it returns an OK response with the response object.

```
public class AuthController {  
  
    @PostMapping("/update/{username}") no usages ▲ 55:18-60:0  
    public ResponseEntity<Map<String, Object>> updateUser(  
        @PathVariable String username,  
        @RequestBody UpdateDto updateDto) {  
  
        Map<String, Object> response = personService.updateUser(username, updateDto);  
  
        if (response.containsKey("error")) {  
            return ResponseEntity.badRequest().body(response);  
        }  
        return ResponseEntity.ok(response);  
    }  
  
}
```

DTOs:

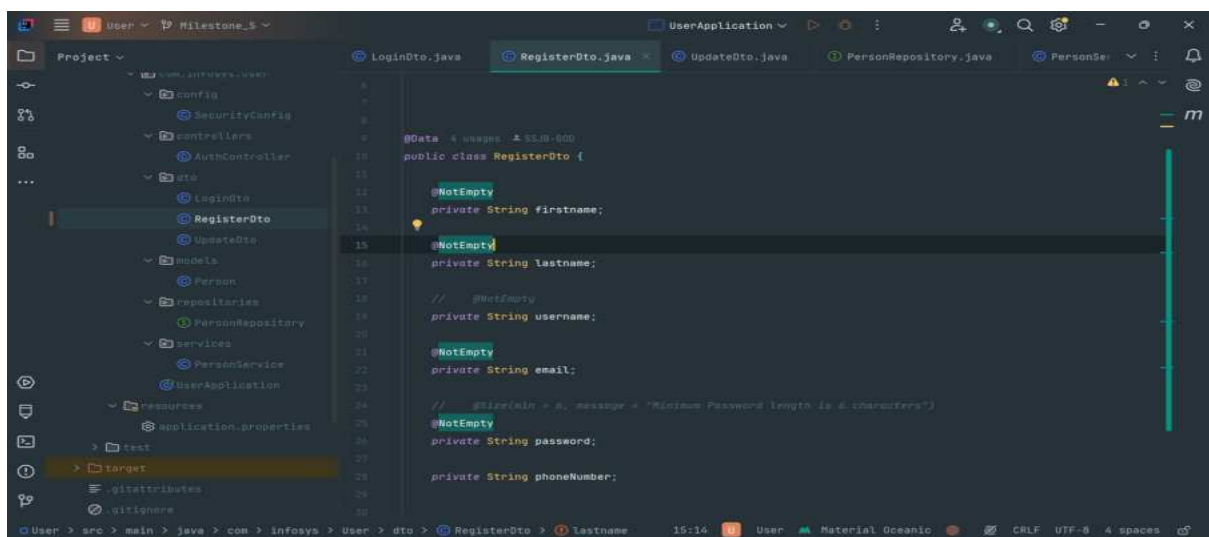
Define DTOs to handle registration data.

Class: LoginDto



```
1 package com.infosys.user.dto;
2
3
4 import lombok.Data;
5
6 @Data @NoArgsConstructor
7 public class LoginDto {
8     private String email;
9     private String password;
10 }
11
```

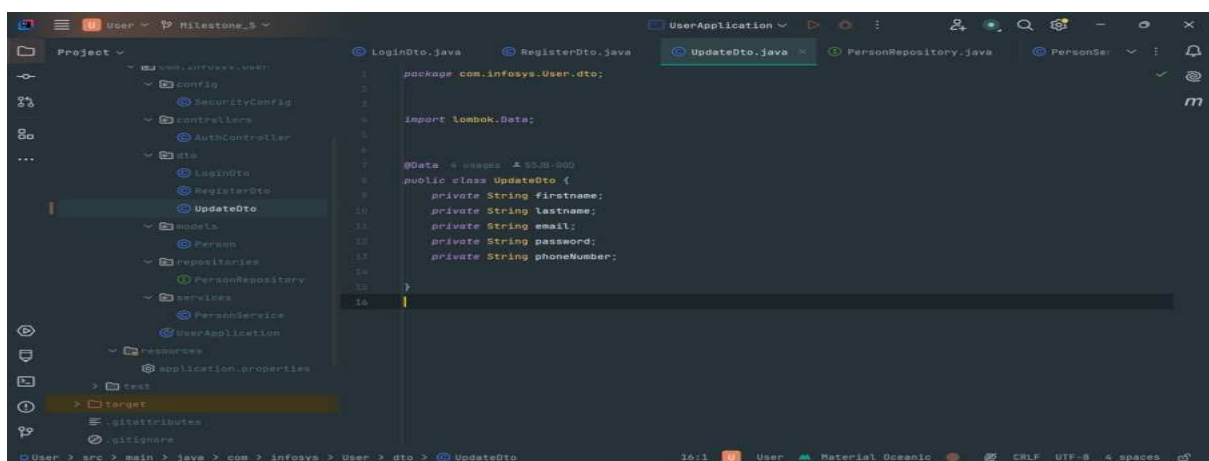
Class: RegisterDto



```
8
9
10 @Data @NoArgsConstructor
11 public class RegisterDto {
12     @NotEmpty
13     private String firstname;
14
15     @NotEmpty
16     private String lastname;
17
18     // @NotEmpty
19     private String username;
20
21     @NotEmpty
22     private String email;
23
24     // @Size(min = 8, message = "Minimum Password length is 8 characters")
25     @NotEmpty
26     private String password;
27
28     private String phoneNumber;
29
30 }

```

Class: UpdateDto

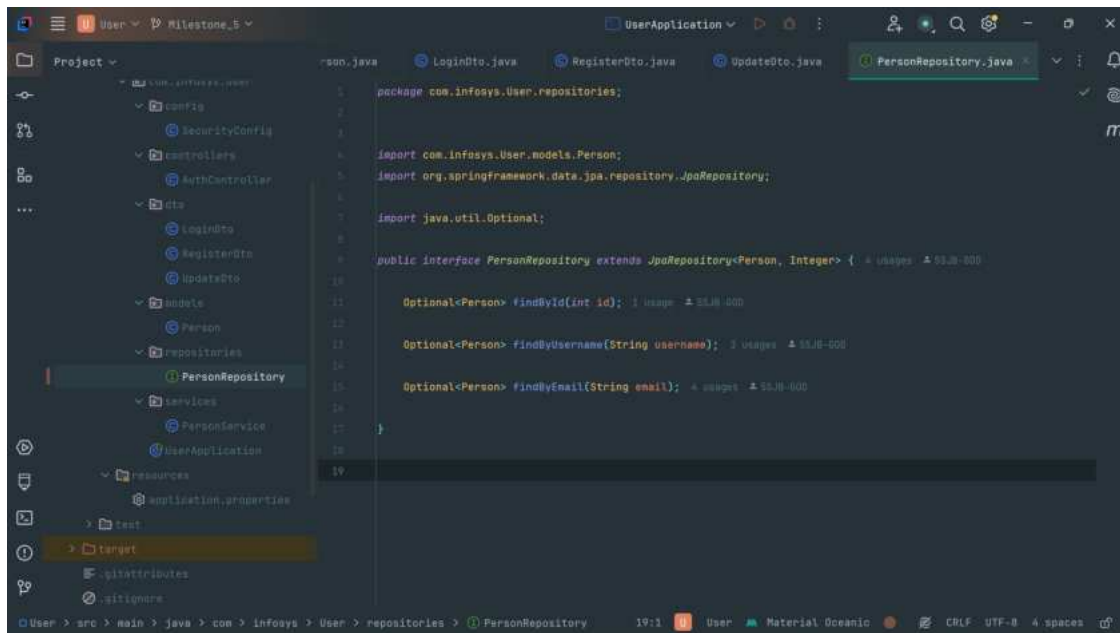


```
1 package com.infosys.user.dto;
2
3
4 import lombok.Data;
5
6 @Data @NoArgsConstructor
7 public class UpdateDto {
8     private String firstname;
9     private String lastname;
10     private String email;
11     private String password;
12     private String phoneNumber;
13 }
14
```

Repository :

Create Repositories for database interactions

Class: PersonRepository

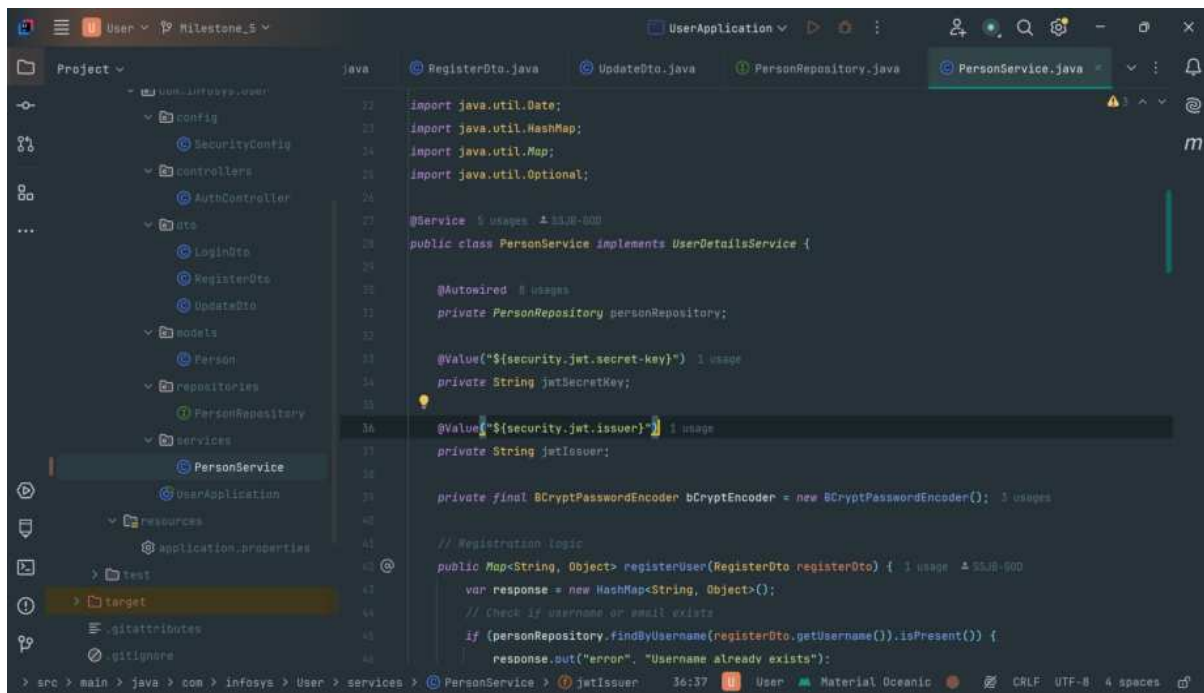


```
1 package com.infosys.user.repositories;
2
3 import com.infosys.user.models.Person;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.Optional;
7
8 public interface PersonRepository extends JpaRepository<Person, Integer> {
9     // 4 usages ▲ 55.18-600
10
11     Optional<Person> findById(int id); // 1 usage ▲ 55.18-600
12
13     Optional<Person> findByUsername(String username); // 1 usage ▲ 55.18-600
14
15     Optional<Person> findByEmail(String email); // 4 usages ▲ 55.18-600
16 }
17
18
19
```

Service:

Create service classes to handle business logic.

Class: PersonService



```
1 import java.util.Date;
2 import java.util.HashMap;
3 import java.util.Map;
4 import java.util.Optional;
5
6 @Service // 5 usages ▲ 55.18-600
7 public class PersonService implements UserDetailsService {
8     // 8 usages
9     @Autowired
10     private PersonRepository personRepository;
11
12     @Value("${security.jwt.secret-key}") // 1 usage
13     private String jwtSecretKey;
14
15     @Value("${security.jwt.issuer}") // 1 usage
16     private String jwtIssuer;
17
18     private final BCryptPasswordEncoder bCryptEncoder = new BCryptPasswordEncoder(); // 3 usages
19
20     // Registration logic
21     public Map<String, Object> registerUser(RegisterDto registerDto) { // 1 usage ▲ 55.18-600
22         var response = new HashMap<String, Object>();
23         // Check if username or email exists
24         if (personRepository.findByUsername(registerDto.getUsername()).isPresent()) {
25             response.put("error", "Username already exists");
26         }
27     }
28 }
29
```


This screenshot shows the `registerUser` method in `PersonService.java`. The method takes a `RegisterDto` and returns a `Map<String, Object>`. It first checks if a user with the same username or email already exists. If not, it creates a new `Person` object, sets its details, encrypts the password, and saves it to the `PersonRepository`. Finally, it generates a JWT token and returns it along with the user details.

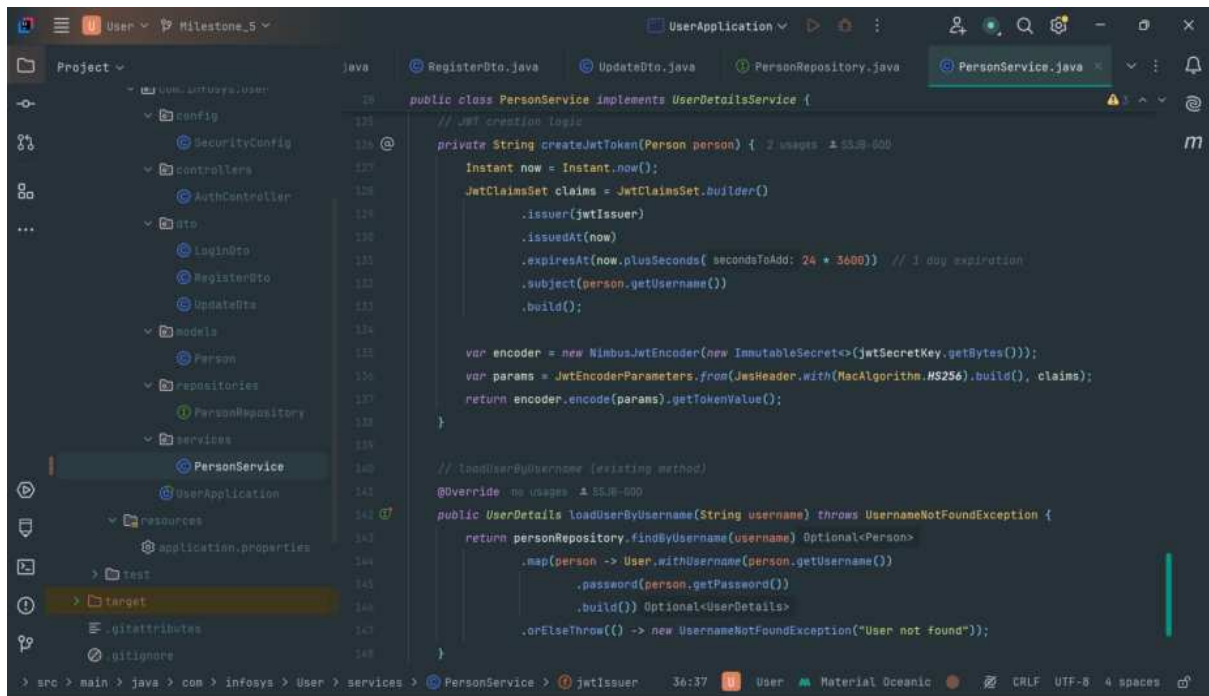
```
public class PersonService implements UserDetailsService {  
    // Check if username or email exists  
    public Map<String, Object> registerUser(RegisterDto registerDto) {  
        if (personRepository.findByUsername(registerDto.getUsername()).isPresent()) {  
            response.put("error", "Username already exists");  
            return response;  
        }  
        if (personRepository.findByEmail(registerDto.getEmail()).isPresent()) {  
            response.put("error", "Email already exists");  
            return response;  
        }  
        // Encrypt password and save user  
        Person person = new Person();  
        person.setFirstname(registerDto.getFirstname());  
        person.setLastname(registerDto.getLastname());  
        person.setUsername(registerDto.getUsername());  
        person.setEmail(registerDto.getEmail());  
        person.setCreatedAt(new Date());  
        person.setPassword(bCryptPasswordEncoder.encode(registerDto.getPassword()));  
        personRepository.save(person);  
  
        response.put("token", createJwtToken(person));  
        response.put("user", person);  
        return response;  
    }  
}
```

This screenshot shows two methods in `PersonService.java`: `authenticateUser` and `updateUser`.
`authenticateUser` takes a `LoginDto` and returns a `Map<String, Object>`. It checks if a user exists with the provided email and password. If correct, it generates a JWT token and returns the user details. Otherwise, it returns an error message.
`updateUser` takes a username and an `UpdateDto`, returning a `Map<String, Object>`. It checks if the user exists. If not, it returns an error. If yes, it updates the user's details (firstname, lastname, email) and returns the updated user details.

```
public Map<String, Object> authenticateUser(LoginDto loginDto) {  
    Optional<Person> personOptional = personRepository.findByEmail(loginDto.getEmail());  
    var response = new HashMap<String, Object>();  
  
    if (personOptional.isPresent()) {  
        Person person = personOptional.get();  
        if (bCryptPasswordEncoder.matches(loginDto.getPassword(), person.getPassword())) {  
            response.put("token", createJwtToken(person));  
            response.put("user", person);  
            return response;  
        }  
    }  
    response.put("error", "Email or password is incorrect");  
    return response;  
}  
  
public Map<String, Object> updateUser(String username, UpdateDto updateDto) {  
    var response = new HashMap<String, Object>();  
  
    Optional<Person> personOptional = personRepository.findByUsername(username);  
    if (personOptional.isEmpty()) {  
        response.put("error", "User not found");  
        return response;  
    }  
}
```

This screenshot shows the `updateUser` method in `PersonService.java`, continuing from the previous screenshot. It updates the user's details based on the `UpdateDto` and returns the updated user details.

```
Person person = personOptional.get();  
  
// Update details  
if (updateDto.getFirstname() != null) {  
    person.setFirstname(updateDto.getFirstname());  
}  
if (updateDto.getLastname() != null) {  
    person.setLastname(updateDto.getLastname());  
}  
if (updateDto.getEmail() != null && !updateDto.getEmail().equals(person.getEmail())) {  
    if (personRepository.findByEmail(updateDto.getEmail()).isPresent()) {  
        response.put("error", "Email already in use");  
        return response;  
    }  
}
```

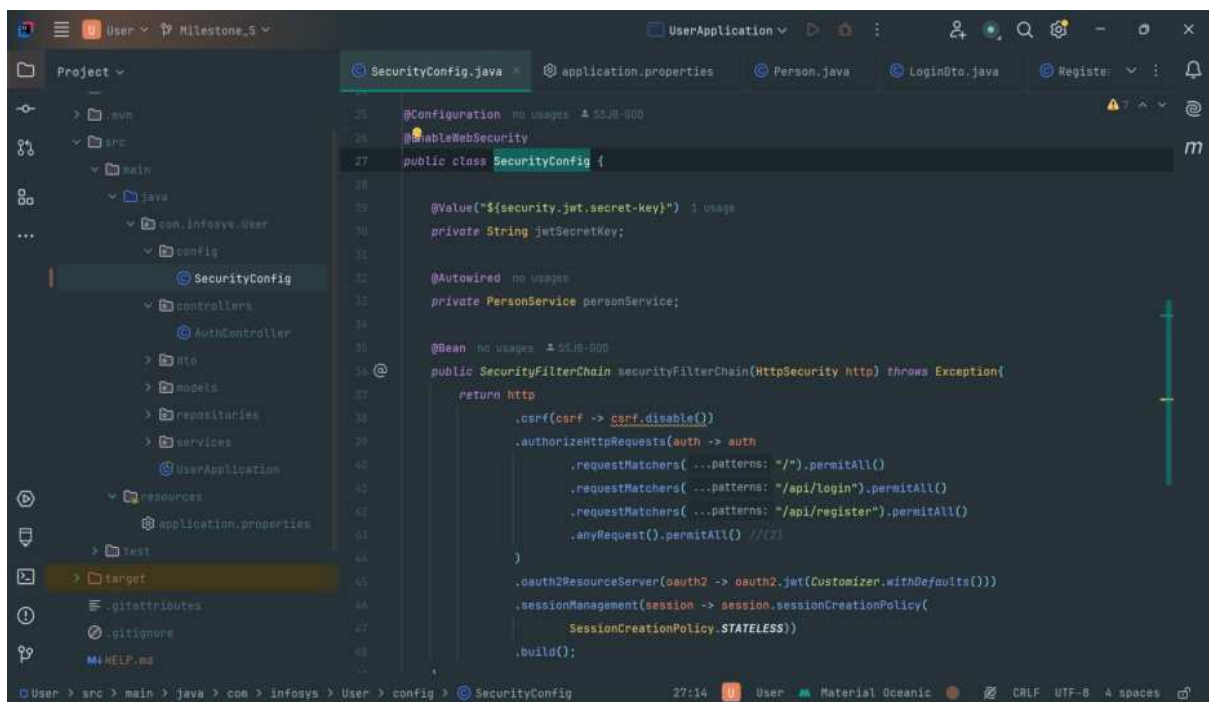


```
129 public class PersonService implements UserDetailsService {
130     // JWT creation logic
131     private String createJwtToken(Person person) {
132         Instant now = Instant.now();
133         JwtClaimsSet claims = JwtClaimsSet.builder()
134             .issuer(jwtIssuer)
135             .issuedAt(now)
136             .expiresAt(now.plusSeconds( secondsToAdd: 24 * 3600)) // 1 day expiration
137             .subject(person.getUsername())
138             .build();
139
140         var encoder = new NimbusJwtEncoder(new ImmutableSecret<>(jwtSecretKey.getBytes()));
141         var params = JwtEncoderParameters.from(JwsHeader.with(HeaderAlgorithm.HS256).build(), claims);
142         return encoder.encode(params).getTokenValue();
143     }
144
145     // loadUserByUsername (existing method)
146     @Override
147     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
148         return personRepository.findByUsername(username).map(person -> User.withUsername(person.getUsername())
149             .password(person.getPassword())
150             .build()) Optional<UserDetails>
151             .orElseThrow(() -> new UsernameNotFoundException("User not found"));
152     }
153 }
```

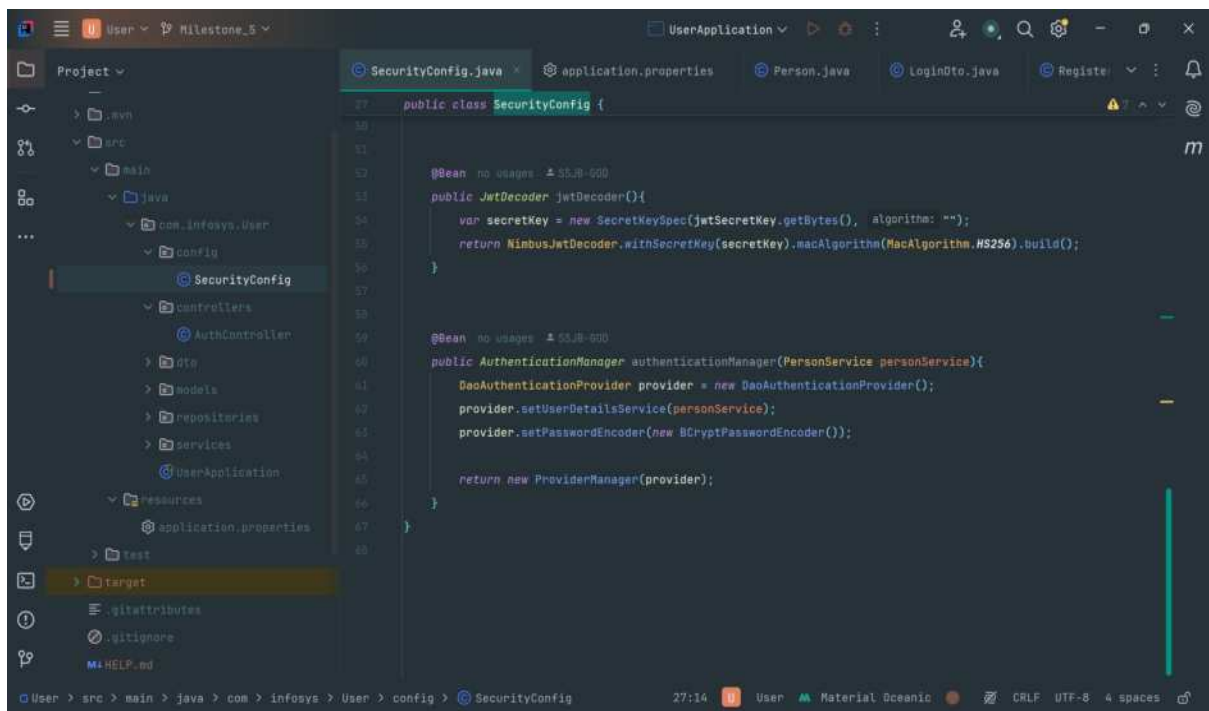
Config:

Create Config for Security Configurations.

Class:SecurityConfig

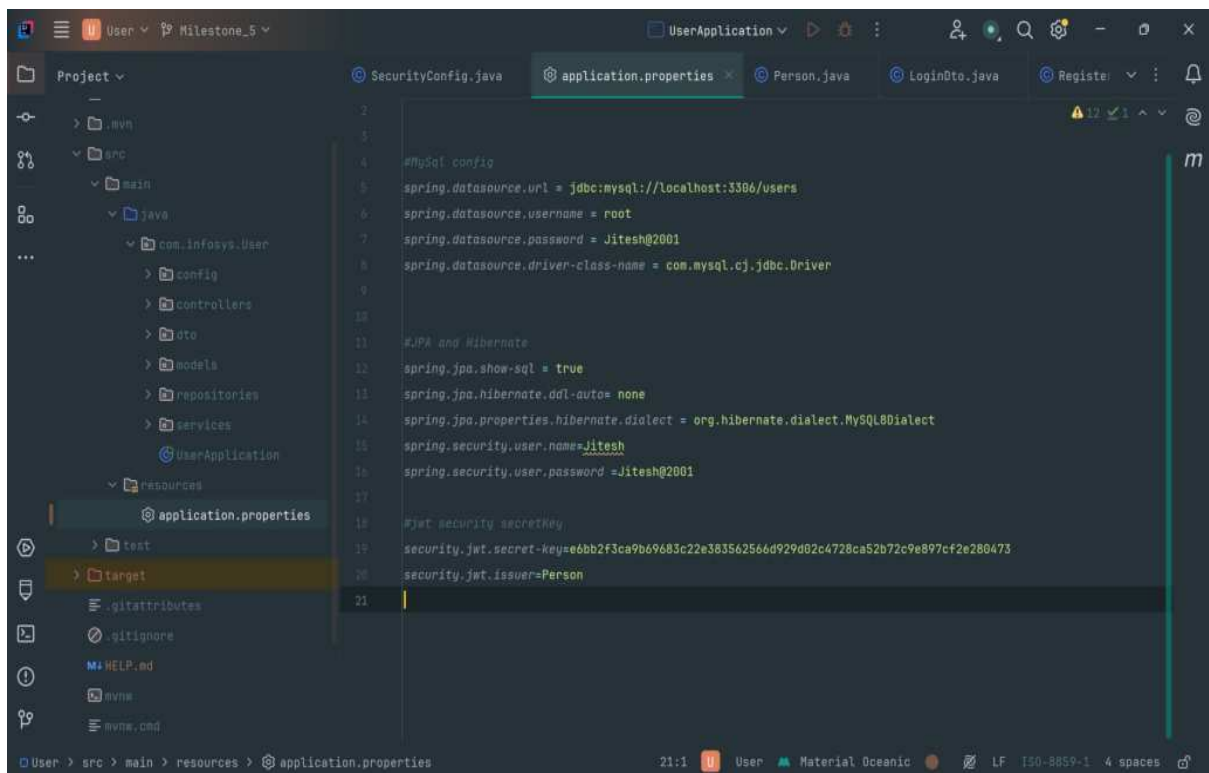


```
25 @Configuration
26 @EnableWebSecurity
27 public class SecurityConfig {
28
29     @Value("${security.jwt.secret-key}")
30     private String jwtSecretKey;
31
32     @Autowired
33     private PersonService personService;
34
35     @Bean
36     public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
37         return http
38             .csrf(csrf -> csrf.disable())
39             .authorizeHttpRequests(auth -> auth
40                 .requestMatchers( ...patterns: "/").permitAll()
41                 .requestMatchers( ...patterns: "/api/login").permitAll()
42                 .requestMatchers( ...patterns: "/api/register").permitAll()
43                 .anyRequest().permitAll() // (2)
44             )
45             .oauth2ResourceServer(oauth2 -> oauth2.jwt(Customizer.withDefaults()))
46             .sessionManagement(session -> session.sessionCreationPolicy(
47                 SessionCreationPolicy.STATELESS))
48             .build();
49     }
50 }
```



```
27 public class SecurityConfig {
28
29
30
31
32
33 @Bean no usages ± 55,18-600
34 public JwtDecoder jwtDecoder(){
35     var secretKey = new SecretKeySpec(jwtSecretKey.getBytes(), algorithm: "");
36     return NimbusJwtDecoder.withSecretKey(secretKey).macAlgorithm(MacAlgorithm.HS256).build();
37 }
38
39
40 @Bean no usages ± 55,18-600
41 public AuthenticationManager authenticationManager(PersonService personService){
42     DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
43     provider.setUserDetailsService(personService);
44     provider.setPasswordEncoder(new BCryptPasswordEncoder());
45
46     return new ProviderManager(provider);
47 }
48 }
```

Application properties

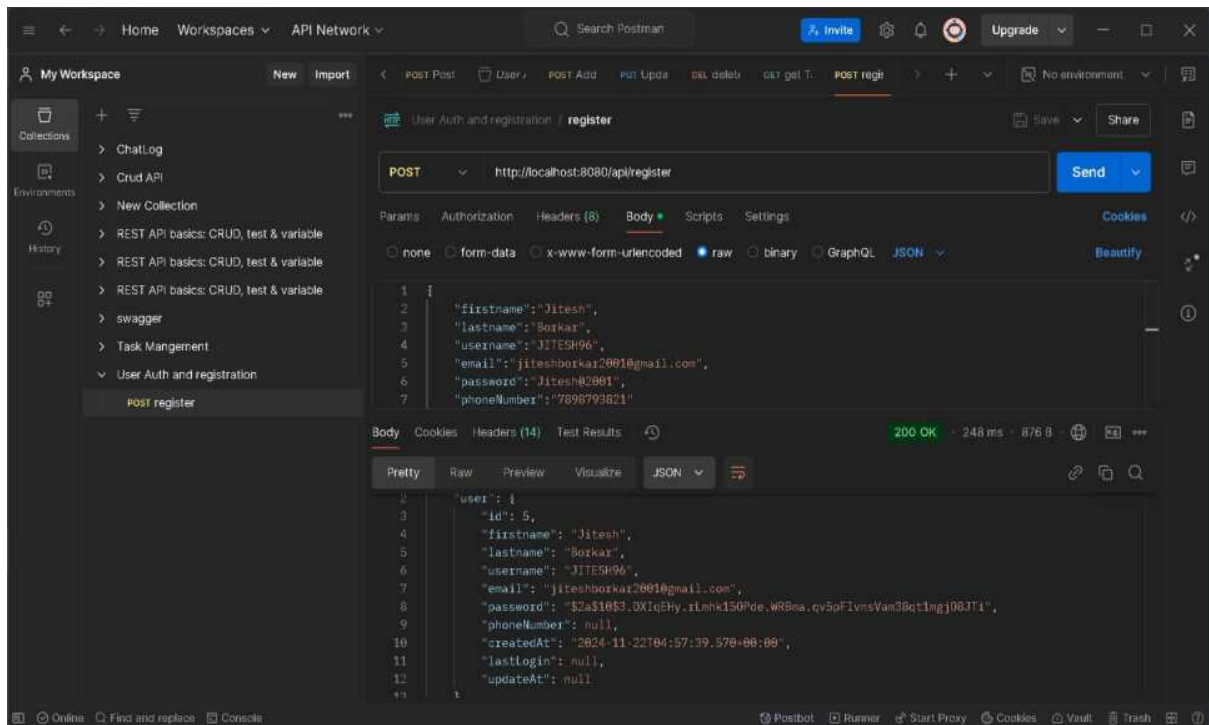


```
2
3
4 #MySQL config
5 spring.datasource.url = jdbc:mysql://localhost:3306/users
6 spring.datasource.username = root
7 spring.datasource.password = Jitesh@2001
8 spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
9
10
11 #JPA and Hibernate
12 spring.jpa.show-sql = true
13 spring.jpa.hibernate.ddl-auto= none
14 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
15 spring.security.user.name=Jitesh
16 spring.security.user.password =Jitesh@2001
17
18 #jwt security secretKey
19 security.jwt.secret-key=e6bb2f3ca9b69683c22e383562566d929d02c4728ca52b72c9e897cf2e280473
20 security.jwt.issuer=Person
21
```

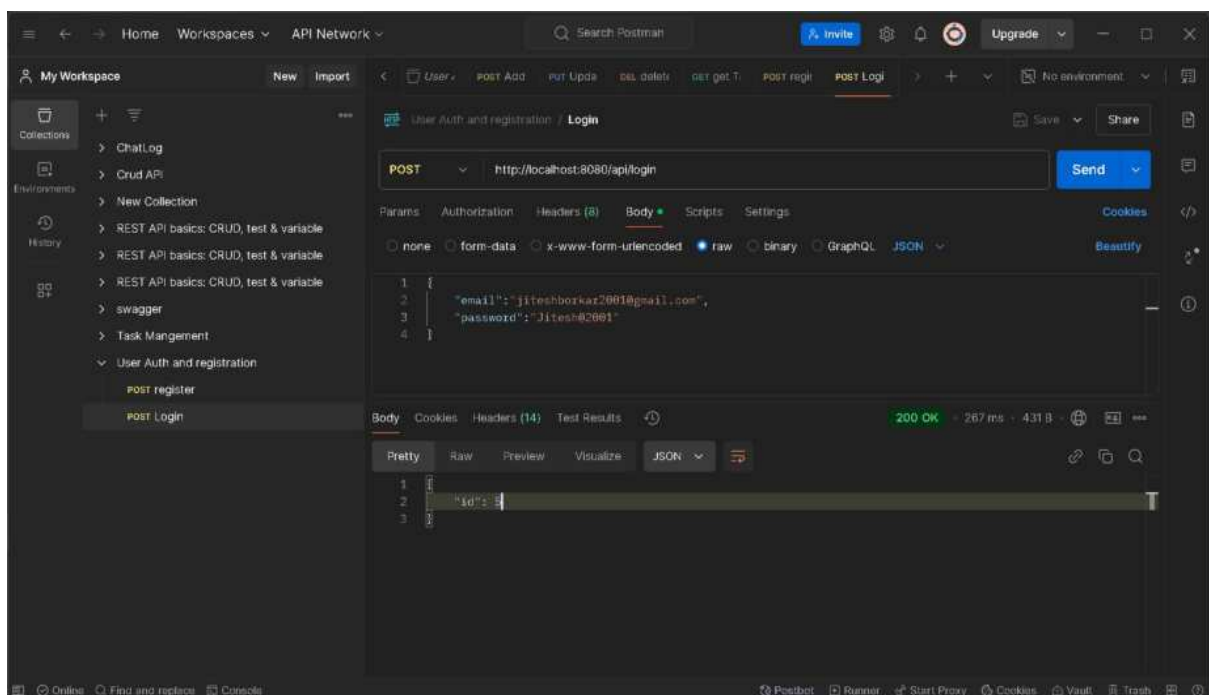

Testing and Validation

Test the application using Postman or other tools to ensure endpoints work as expected.

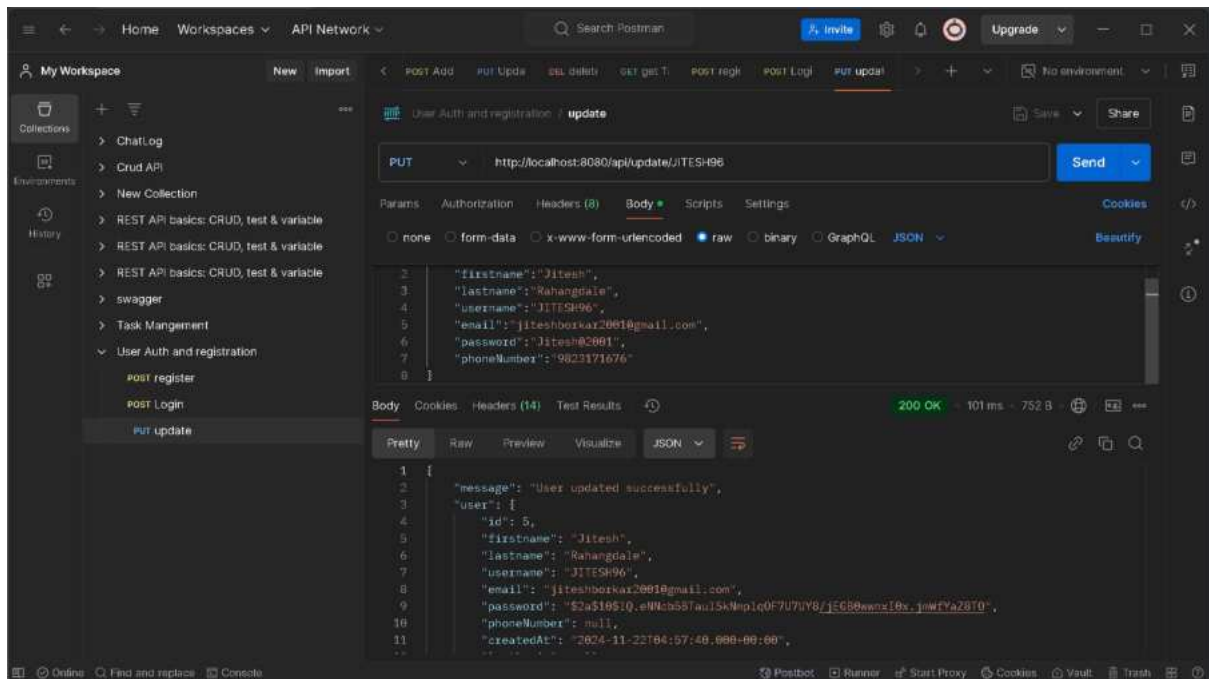
POST api/register



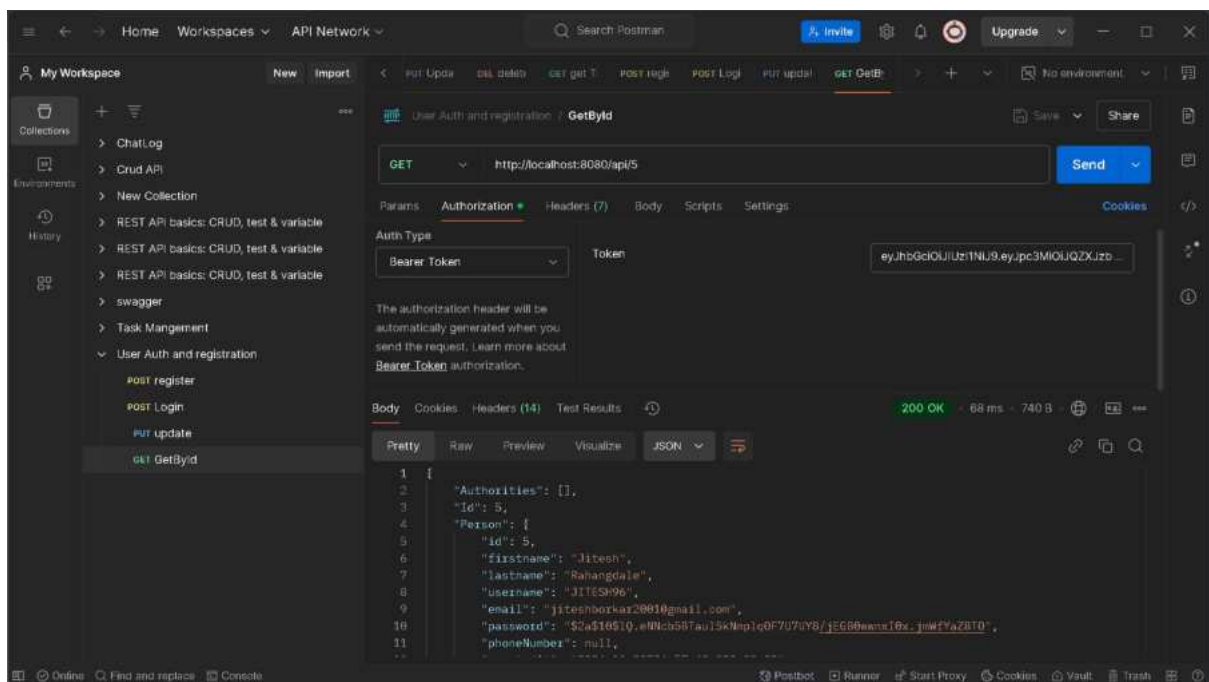
POST api/login



PUT api/update/{username}



GET api/{id}



Conclusion:

This outline provides a foundational structure for a backend for User Authentication and Registration.

Milestone – 2:

The Assessment module evaluates a user's mental health by prompting them to respond to depression-related questions. Based on their answers, the system utilizes a scoring algorithm to assess their mental health status. Once the score is generated, the user receives personalized recommendations and resources tailored to their specific needs ensuring users have access to relevant and helpful guidance for improving their well-being.

Dependencies:

- Lombok
- spring data JPA
- Spring data MongoDB
- Spring web
- MySQL driver

Packages & Classes:

- Controller
 - QuestionController
 - WellbeingTestController
- DTO
 - WellbeingTestDTO
- Exception
 - GlobalExceptionHandler
- Model
 - Choice
 - Question
 - WellbeingTest
- Repository
 - QuestionRepository
 - WellbeingTestRepo
- Service
 - QuestionService
 - WellbeingService
 - WellbeingServiceImpl

Model :

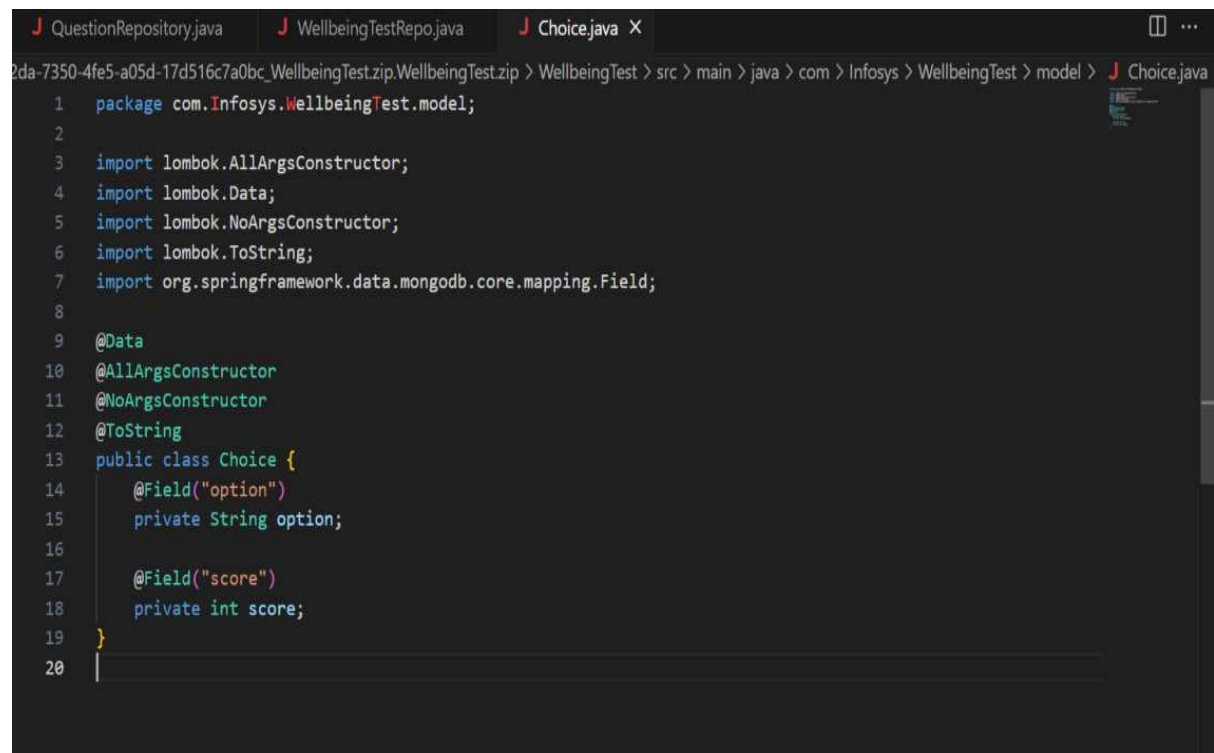
User: Represents a user taking the assessment.

Questionnaire: Stores questions and options.

Assessment: Tracks the results of the assessment.

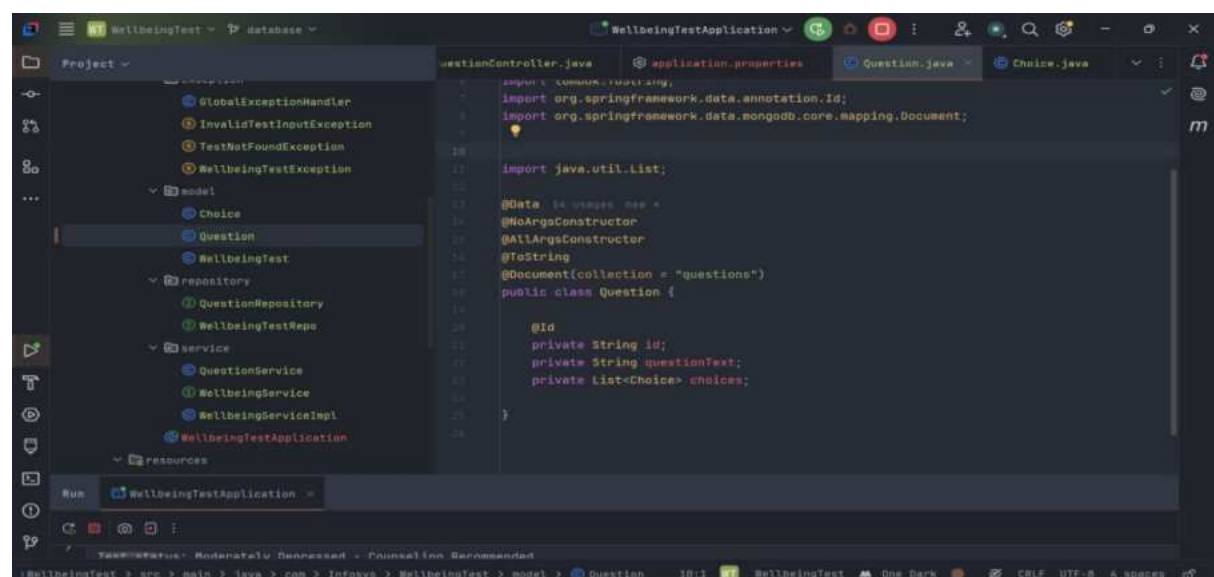
TestHistory: Logs completed assessments for each user.

Class: Choice



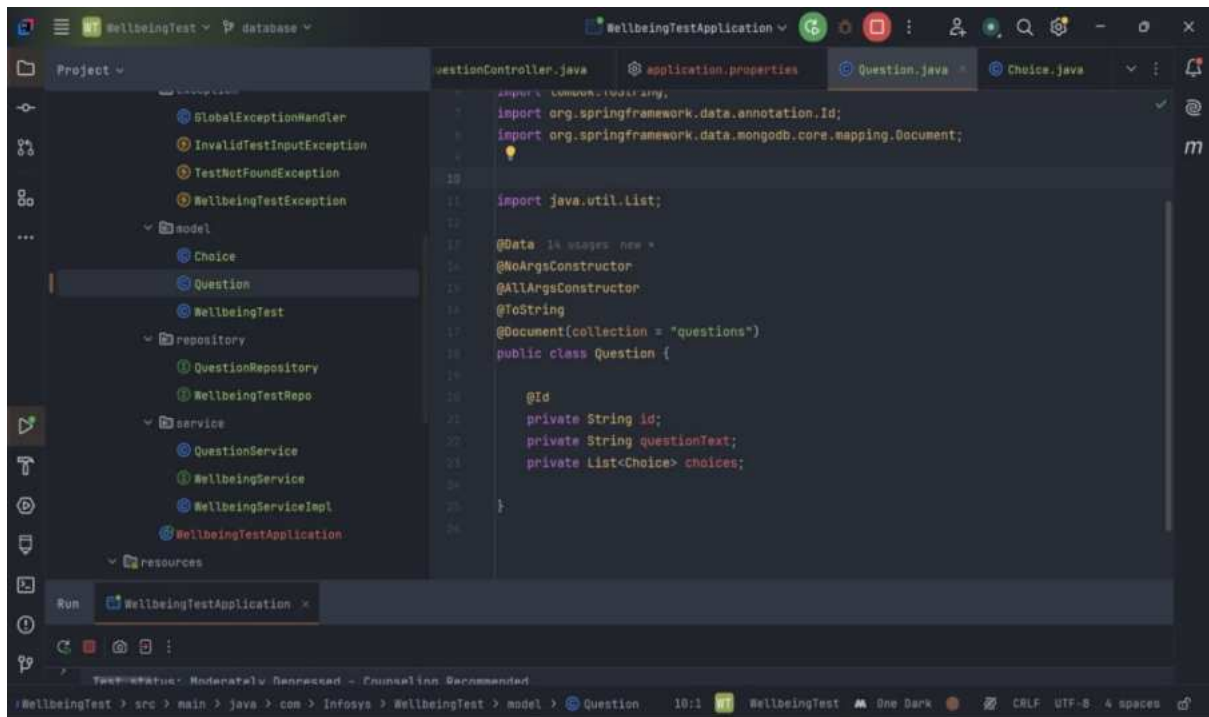
```
1 package com.Infosys.WellbeingTest.model;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
7 import org.springframework.data.mongodb.core.mapping.Field;
8
9 @Data
10 @AllArgsConstructor
11 @NoArgsConstructor
12 @ToString
13 public class Choice {
14     @Field("option")
15     private String option;
16
17     @Field("score")
18     private int score;
19 }
20
```

Class : Question



```
1 import org.springframework.data.annotation.Id;
2 import org.springframework.data.mongodb.core.mapping.Document;
3
4 import java.util.List;
5
6 @Data
7 @NoArgsConstructor
8 @AllArgsConstructor
9 @ToString
10 @Document(collection = "questions")
11 public class Question {
12
13     @Id
14     private String id;
15     private String questionText;
16     private List<Choice> choices;
17 }
18
```

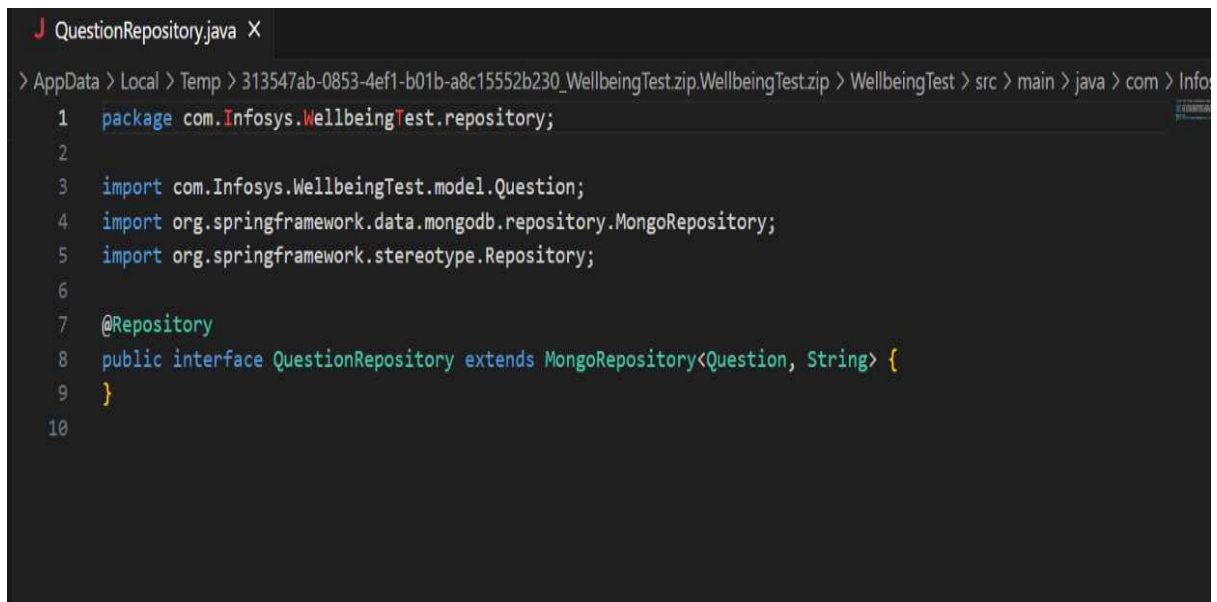
Class : WellbeingTest



Repository :

Define JPA repositories for each entity

Class: QuestionRepository



Class: WellbeingTestRepo

```
QuestionRepository.java WellbeingTestRepo.java X
-be67-4d5afab1ffbe_WellbeingTest.zip.WellbeingTest.zip > WellbeingTest > src > main > java > com > Infosys > WellbeingTest > repository > J WellbeingTestRepo.java

1 package com.Infosys.WellbeingTest.repository;
2
3 import com.Infosys.WellbeingTest.model.WellbeingTest;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.stereotype.Repository;
6
7 import java.util.List;
8 import java.util.Optional;
9
10 @Repository
11 public interface WellbeingTestRepo extends JpaRepository<WellbeingTest, Integer> {
12     Optional<WellbeingTest> findById(int id);
13     List<WellbeingTest> findByUserId(int user_id);
14 }
15
```

Service:

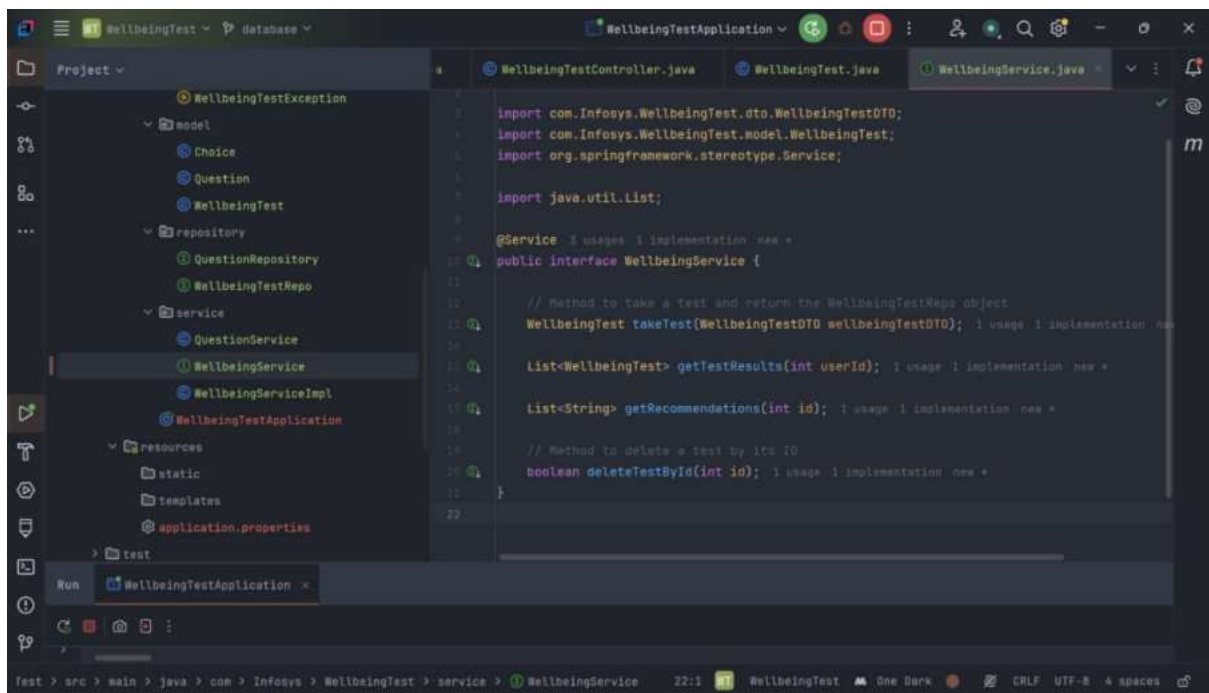
Create service classes to handle business logic, such as saving answers, calculating assessment scores, and logging test history.

Class: QuestionService

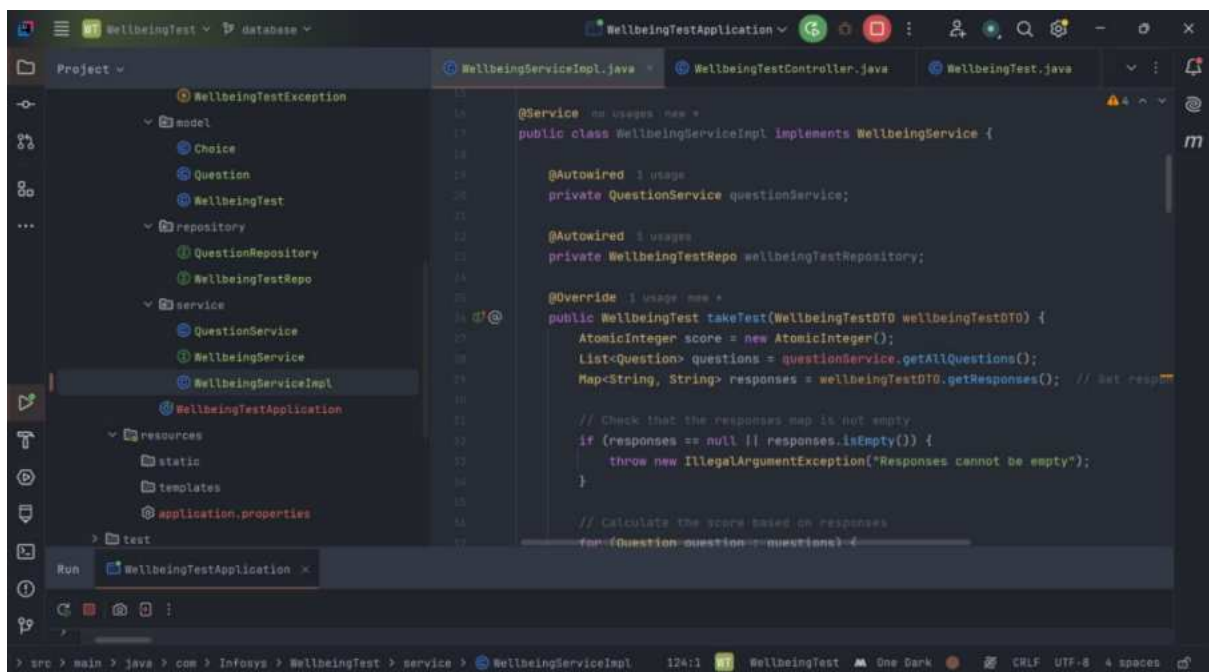
```
QuestionRepository.java WellbeingTestRepo.java Choice.java QuestionService.java X
4b60-9720-9d3bf657383d_WellbeingTest.zip.WellbeingTest.zip > WellbeingTest > src > main > java > com > Infosys > WellbeingTest > service > J QuestionService

1 package com.Infosys.WellbeingTest.service;
2
3 import com.Infosys.WellbeingTest.model.Question;
4 import com.Infosys.WellbeingTest.repository.QuestionRepository;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.stereotype.Service;
7
8 import java.util.List;
9
10 @Service
11 public class QuestionService {
12
13     @Autowired
14     private QuestionRepository questionRepository;
15
16     public List<Question> getAllQuestions() {
17         return questionRepository.findAll();
18     }
19
20     public Question getQuestionById(String id) {
21         return questionRepository.findById(id).orElse(null);
22     }
23
24     public Question saveQuestion(Question question) {
25         return questionRepository.save(question);
26     }
27 }
28
```

Class: WellbeingService



Class: WellbeignServiceImpl



```
17 public class WellbeingServiceImpl implements WellbeingService {
18     public WellbeingTest takeTest(WellbeingTestDTO wellbeingTestDTO) {
19         // Calculate the score based on responses
20         for (Question question : questions) {
21             String questionText = question.getQuestionText();
22             String answer = responses.get(questionText); // Get user's response for the question
23
24             if (answer != null && !answer.isEmpty()) {
25                 List<Choice> choices = question.getChoices(); // Get available choices
26
27                 if (choices != null && !choices.isEmpty()) {
28                     Optional<Choice> selectedChoice = choices.stream()
29                         .filter(choice -> choice.getOption().equalsIgnoreCase(answer))
30                         .findFirst();
31
32                     // Add the score for the selected choice
33                     selectedChoice.ifPresent(choice -> {
34                         score.addAndGet(choice.getScore());
35                         System.out.println("Added score for question: " + questionText);
36                     });
37                 } else {
38                     System.out.println("No choices available for question: " + questionText);
39                 }
40             }
41         }
42
43         // Log the final score to verify
44         System.out.println("Final calculated score: " + score);
45
46         // Create a WellbeingTest object to save to the database
47         WellbeingTest wellbeingTest = new WellbeingTest();
48         wellbeingTest.setUserId(wellbeingTestDTO.getUserId()); // Set the user_id
49         wellbeingTest.setScore(score.get());
50         wellbeingTest.setTaken_at(LocalDateTime.now());
51         wellbeingTest.setStatus(determineStatus(score.get()));
52
53         // Save the wellbeingTest object to the repository
54         return wellbeingTestRepository.save(wellbeingTest);
55     }
56
57     @Override // Usage: new
58     public List<WellbeingTest> getTestResults(int userId) {
59         // TODO: Implement this method
60     }
61 }
```

```
17 public class WellbeingTestController {
18     private WellbeingService wellbeingService;
19
20     public WellbeingTest takeTest(WellbeingTestDTO wellbeingTestDTO) {
21         return wellbeingService.takeTest(wellbeingTestDTO);
22     }
23
24     // Log the final score to verify
25     System.out.println("Final calculated score: " + score);
26
27     // Create a WellbeingTest object to save to the database
28     WellbeingTest wellbeingTest = new WellbeingTest();
29     wellbeingTest.setUserId(wellbeingTestDTO.getUserId()); // Set the user_id
30     wellbeingTest.setScore(score.get());
31     wellbeingTest.setTaken_at(LocalDateTime.now());
32     wellbeingTest.setStatus(determineStatus(score.get()));
33
34     // Save the wellbeingTest object to the repository
35     return wellbeingTestRepository.save(wellbeingTest);
36 }
37
38 @Override // Usage: new
39 public List<WellbeingTest> getTestResults(int userId) {
40     // TODO: Implement this method
41 }
```

```
17 public class WellbeingTestApplication {
18     public static void main(String[] args) {
19         SpringApplication.run(WellbeingTestApplication.class, args);
20     }
21
22     // TODO: Implement this method
23 }
24
25 // Run the application
26 // Run the application
27 // Run the application
28 // Run the application
29 // Run the application
30 // Run the application
31 // Run the application
32 // Run the application
33 // Run the application
34 // Run the application
35 // Run the application
36 // Run the application
37 // Run the application
38 // Run the application
39 // Run the application
40 // Run the application
41 // Run the application
42 // Run the application
43 // Run the application
44 // Run the application
45 // Run the application
46 // Run the application
47 // Run the application
48 // Run the application
49 // Run the application
50 // Run the application
51 // Run the application
52 // Run the application
53 // Run the application
54 // Run the application
55 // Run the application
56 // Run the application
57 // Run the application
58 // Run the application
59 // Run the application
60 // Run the application
61 // Run the application
62 // Run the application
63 // Run the application
64 // Run the application
65 // Run the application
66 // Run the application
67 // Run the application
68 // Run the application
69 // Run the application
70 // Run the application
71 // Run the application
72 // Run the application
73 // Run the application
74 // Run the application
75 // Run the application
76 // Run the application
77 // Run the application
78 // Run the application
79 // Run the application
80 // Run the application
81 // Run the application
82 // Run the application
83 // Run the application
84 // Run the application
85 // Run the application
86 // Run the application
87 // Run the application
88 // Run the application
89 // Run the application
90 // Run the application
91 // Run the application
92 // Run the application
93 // Run the application
94 // Run the application
95 // Run the application
96 // Run the application
97 // Run the application
98 // Run the application
99 // Run the application
100 // Run the application
```

Controller :

Create REST controllers to handle API requests, like saving answers, retrieving questionnaires, and fetching test histories.

Class: QuestionController

```
QuestionRepository.java WellbeingTestRepo.java Choice.java QuestionService.java
308-b5e9e1ff1ab6_WellbeingTest.zip.WellbeingTest.zip > WellbeingTest > src > main > java > com > Infosys > WellbeingT
1 package com.Infosys.WellbeingTest.controller;
2
3 import com.Infosys.WellbeingTest.model.Question;
4 import com.Infosys.WellbeingTest.service.QuestionService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.*;
7
8 import java.util.List;
9
10 @RestController
11 @RequestMapping("/api/questions")
12 public class QuestionController {
13
14     @Autowired
15     private QuestionService questionService;
16
17     @GetMapping
18     public List<Question> getAllQuestions() {
19         return questionService.getAllQuestions();
20     }
21
22     @PostMapping
23     public Question addQuestion(@RequestBody Question question) {
24         return questionService.saveQuestion(question);
25     }
26 }
27
```

Class: WellbeingTestController

```
WellbeingTestRepo.java Choice.java QuestionService.java QuestionController.java WellbeingTestController.java
-fe77e2b81a99_WellbeingTest.zip.WellbeingTest.zip > WellbeingTest > src > main > java > com > Infosys > WellbeingTest > controller > WellbeingTestContro
1 package com.Infosys.WellbeingTest.controller;
2
3 import com.Infosys.WellbeingTest.dto.WellbeingTestDTO;
4 import com.Infosys.WellbeingTest.model.WellbeingTest;
5 import com.Infosys.WellbeingTest.service.WellbeingService;
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.*;
10
11 import java.util.List;
12
13 @RestController
14 @RequestMapping("/api")
15 public class WellbeingTestController {
16
17     @Autowired
18     private WellbeingService wellbeingService;
19
20     @PostMapping("/takeTest")
21     public ResponseEntity<WellbeingTest> takeTest(@RequestBody WellbeingTestDTO wellbeingTestDTO) {
22         WellbeingTest result = wellbeingService.takeTest(wellbeingTestDTO);
23         return ResponseEntity.status(HttpStatus.CREATED).body(result);
24     }
25
26     @GetMapping("/getTestResults/{userId}")
27     public List<WellbeingTest> getTestResults(@PathVariable int userId) {
28         return wellbeingService.getTestResults(userId);
29     }
30 }
```

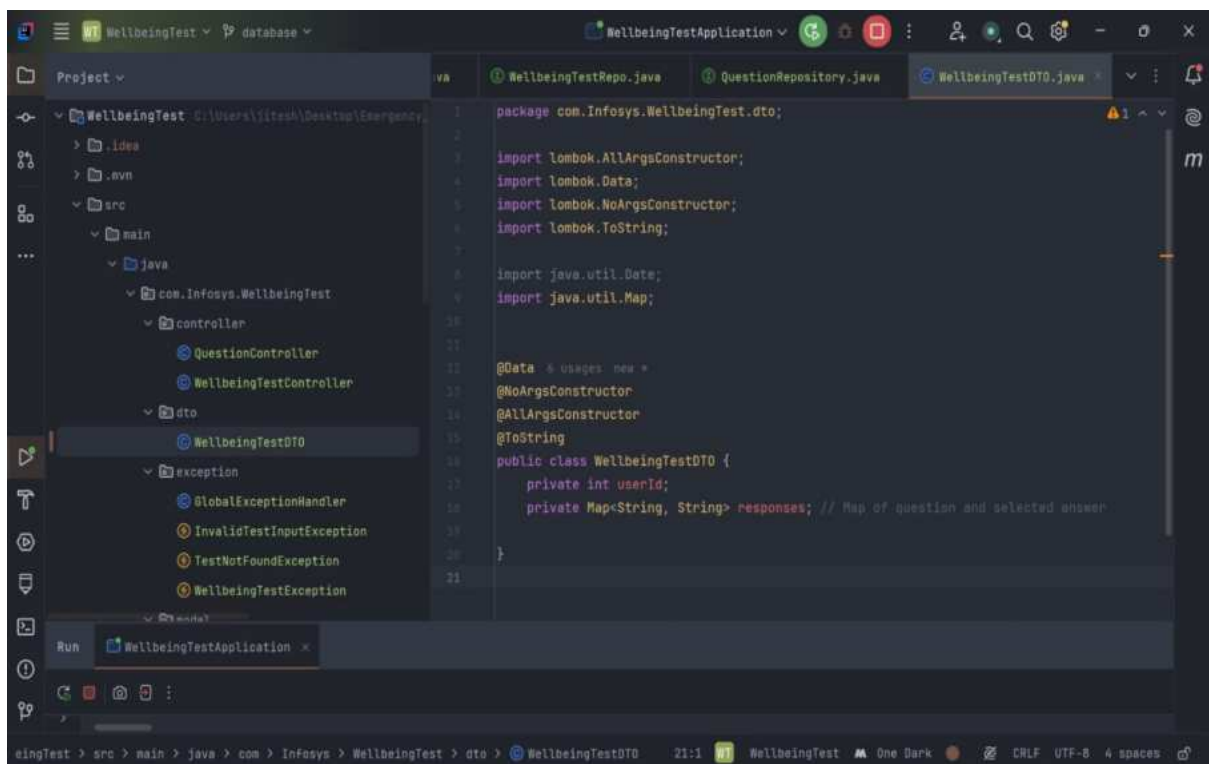


```
WellbeingTestRepo.java | Choice.java | QuestionService.java | QuestionController.java | WellbeingTestController.java
fe77e2b81a99_WellbeingTest.zip.WellbeingTest.zip > WellbeingTest > src > main > java > com > Infosys > WellbeingTest > controller > WellbeingTestController.java
15 public class WellbeingTestController {
16     @GetMapping("/getTestResults/{id}")
17     public List<WellbeingTest> getTestResults(@PathVariable int id) {
18         return wellbeingService.getTestResults(id);
19     }
20
21     @GetMapping("/getRecommendations/{id}")
22     public List<String> getRecommendations(@PathVariable int id) {
23         return wellbeingService.getRecommendations(id);
24     }
25
26     // Endpoint to delete a test by ID (DELETE request)
27     @DeleteMapping("/deleteTest/{id}")
28     public ResponseEntity<String> deleteTestById(@PathVariable("id") int id) {
29         // Call the service to delete the test
30         boolean isDeleted = wellbeingService.deleteTestById(id);
31
32         // Return appropriate response based on whether the deletion was successful
33         if (isDeleted) {
34             return new ResponseEntity<>("Test deleted successfully.", HttpStatus.OK);
35         } else {
36             return new ResponseEntity<>("Test not found.", HttpStatus.NOT_FOUND);
37         }
38     }
39 }
40
41
42
43
44
45
46
47
48
49
50
51
```

DTO :

DTOs will help ensure that only the necessary information is sent to the client

Class : WellbeingTestDTO



The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with folders for `.idea`, `.svn`, `src`, and `main`. The `main` folder contains `java`, `com`, and `exception`. The `com` folder contains `Infosys`, which contains `WellbeingTest`. The `WellbeingTest` folder contains `controller` (with `QuestionController` and `WellbeingTestController`), `dto` (with `WellbeingTestDTO`), and `exception` (with `GlobalExceptionHandler`, `InvalidTestInputException`, `TestNotFoundException`, and `WellbeingTestException`).
- Code Editor:** Displays the `WellbeingTestDTO.java` file with the following code:

```
package com.Infosys.WellbeingTest.dto;

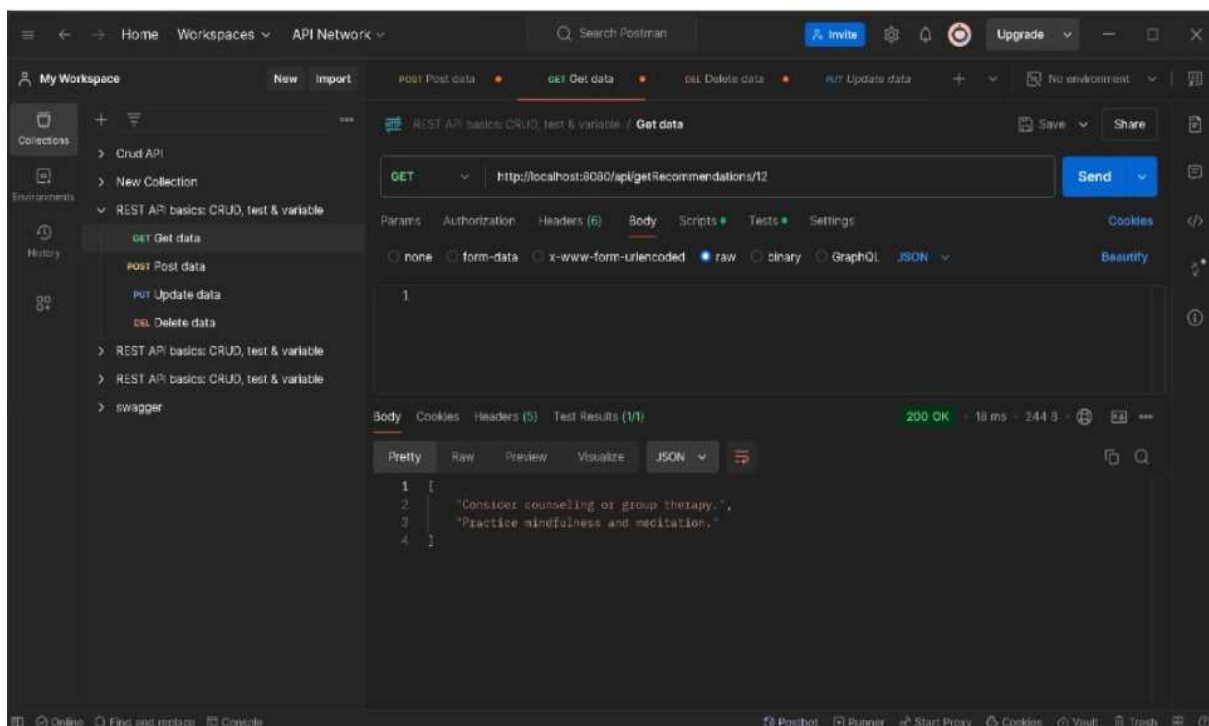
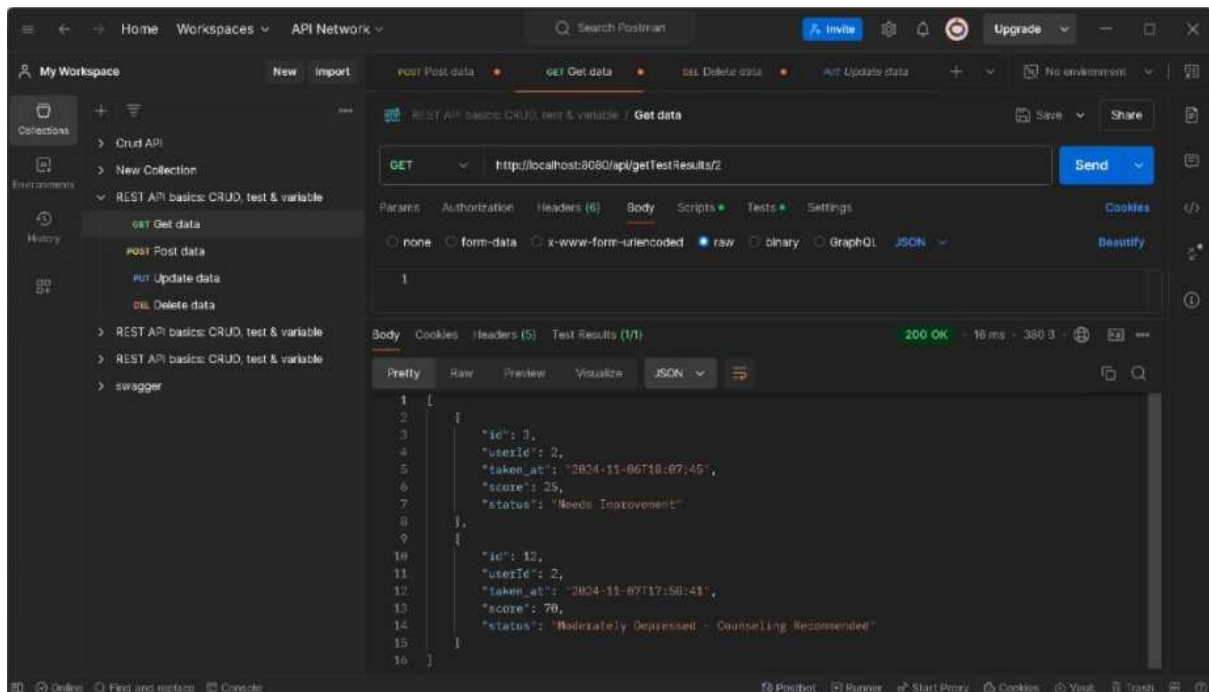
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

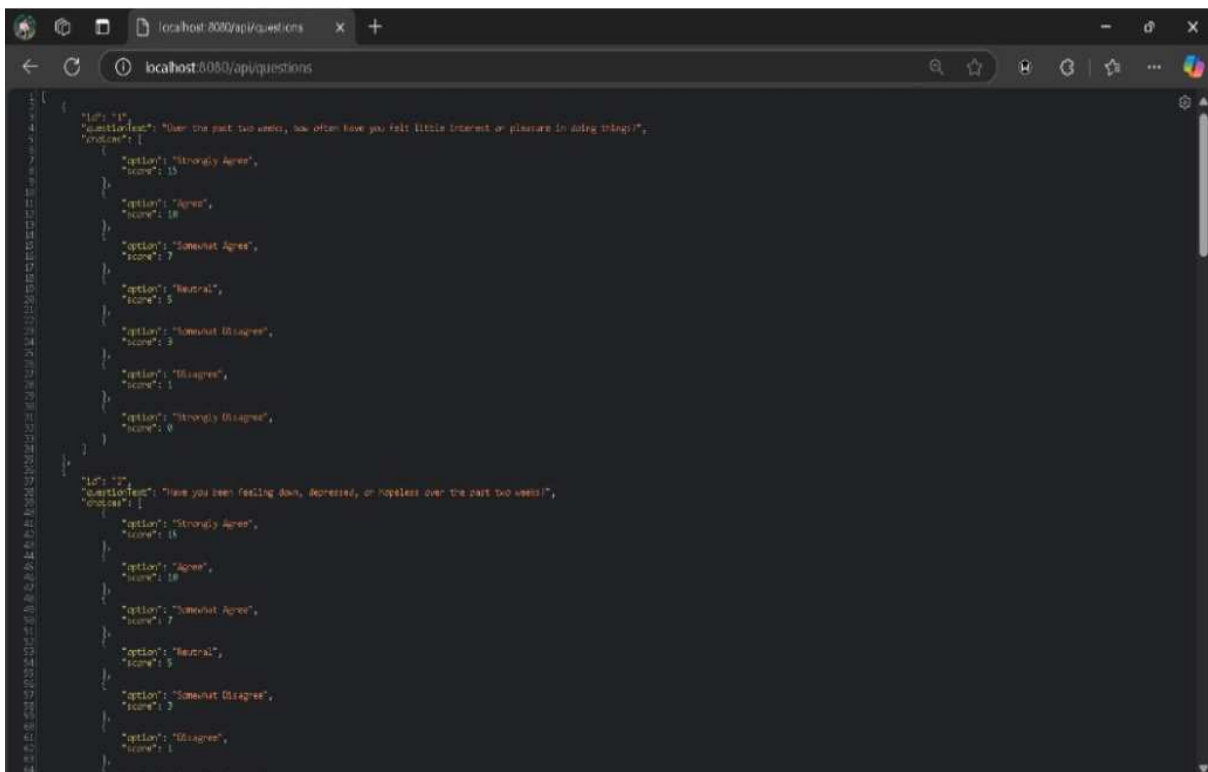
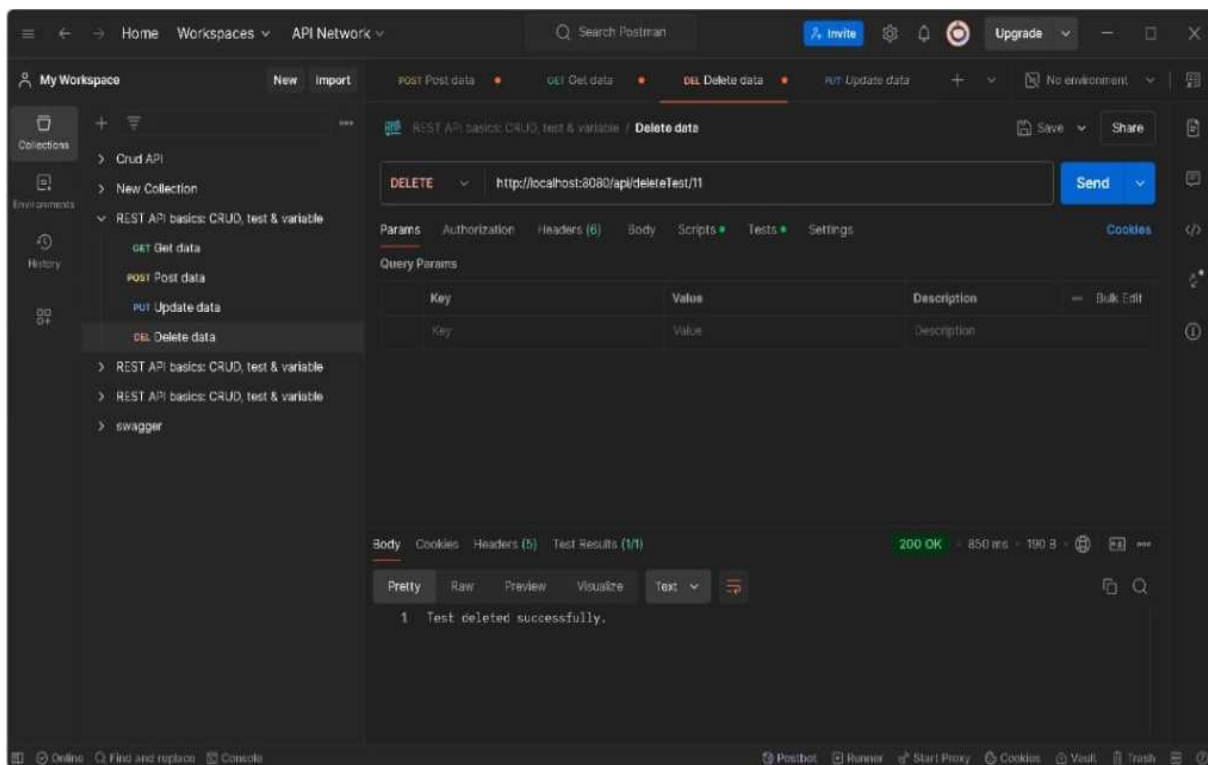
import java.util.Date;
import java.util.Map;

@Data & uses = new *
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class WellbeingTestDTO {
    private int userId;
    private Map<String, String> responses; // Map of question and selected answer
}
```
- Run Configuration:** Shows a configuration for `WellbeingTestApplication`.
- Status Bar:** Shows the file path `WellbeingTest > src > main > java > com > Infosys > WellbeingTest > dto > WellbeingTestDTO` and the file encoding `UTF-8`.

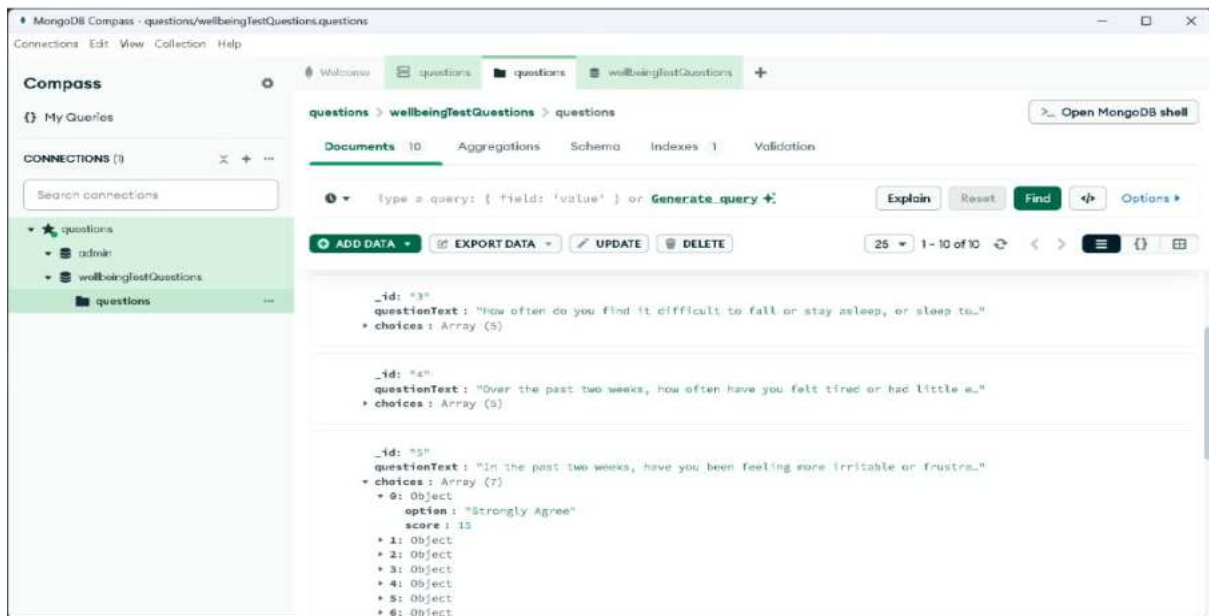
Testing and Validation

Test the application using Postman or other tools to ensure endpoints work as expected.

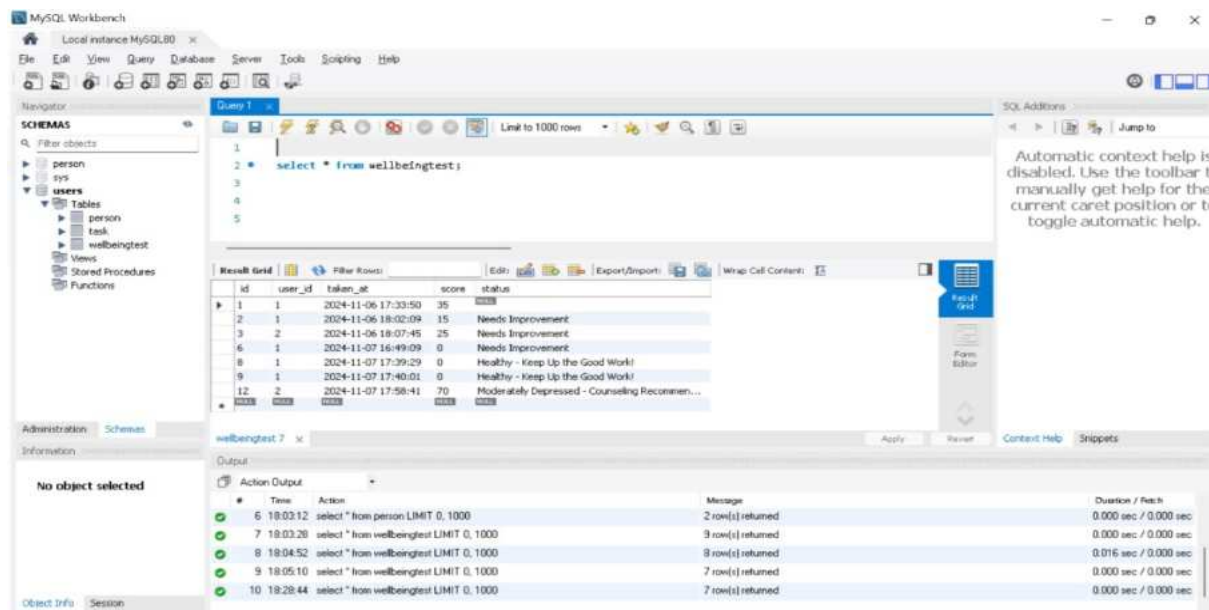




MongoDB



MySQL



Conclusion:

This outline provides a foundational structure for a backend supporting a mental health questionnaire and assessment system.

Milestone – 3:(module 4)

The chatbot offers real-time assistance and answers to users' queries about safety and wellness. It is designed to provide immediate responses and guide users through various features of the application.

Dependencies:

→Lombok

→spring data JPA

→Spring data MongoDB

→Spring web

→MySQL driver

Packages & Classes:

→Controller

- ChatLogController

→DTO

- MessageRequestDTO

→Exception

- GlobalExceptionHandler

→Model

- ChatLog
- MongoChatLog

→Repository

- ChatLogRepository
- MongoChatLogRepository

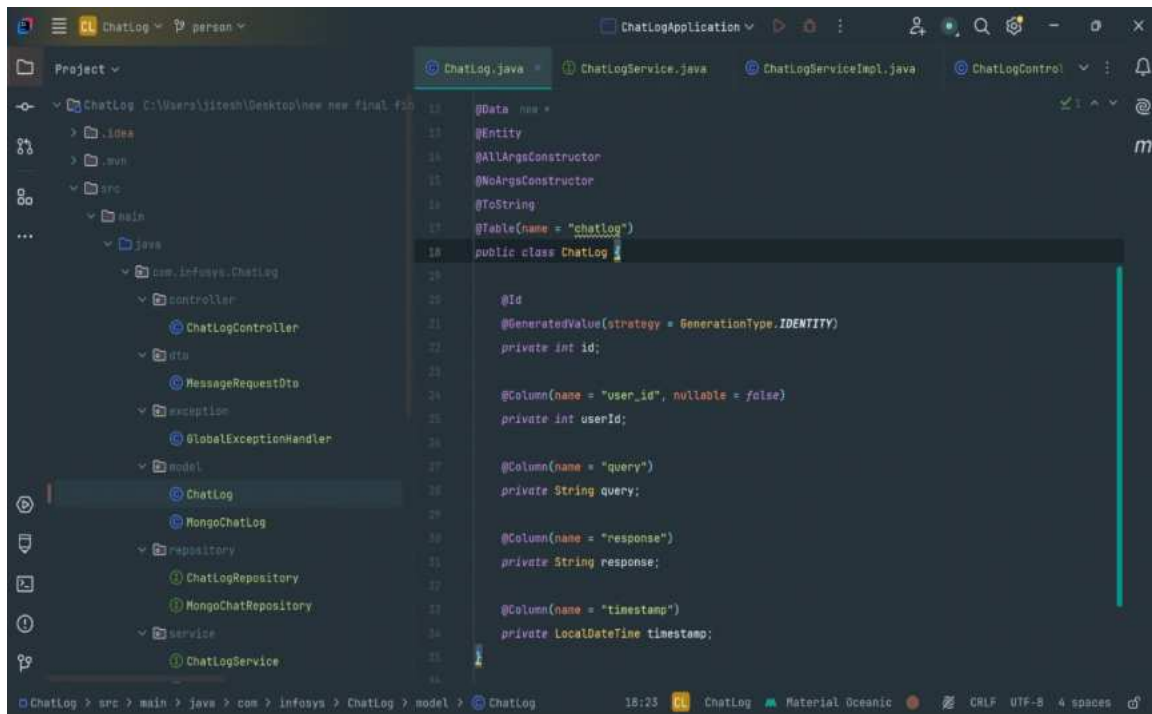
→Service

- ChatLogService
- ChatLogServiceImpl

Model :

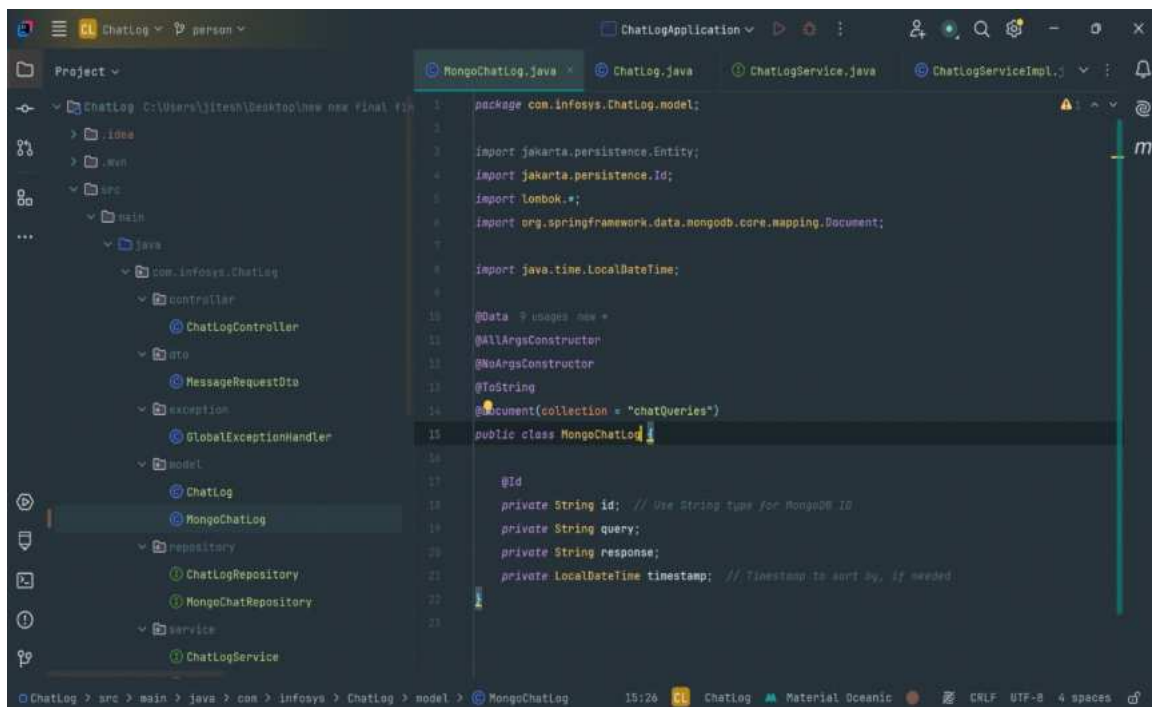
Define models to represent chatbot's messages.

Class: ChatLog



```
1  @Data
2  @Entity
3  @GeneratedValue(strategy = GenerationType.IDENTITY)
4  @Table(name = "chatlog")
5  public class ChatLog {
6
7      @Id
8      @GeneratedValue(strategy = GenerationType.IDENTITY)
9      private int id;
10
11      @Column(name = "user_id", nullable = false)
12      private int userId;
13
14      @Column(name = "query")
15      private String query;
16
17      @Column(name = "response")
18      private String response;
19
20      @Column(name = "timestamp")
21      private LocalDateTime timestamp;
22
23  }
```

Class : MongoChatLog

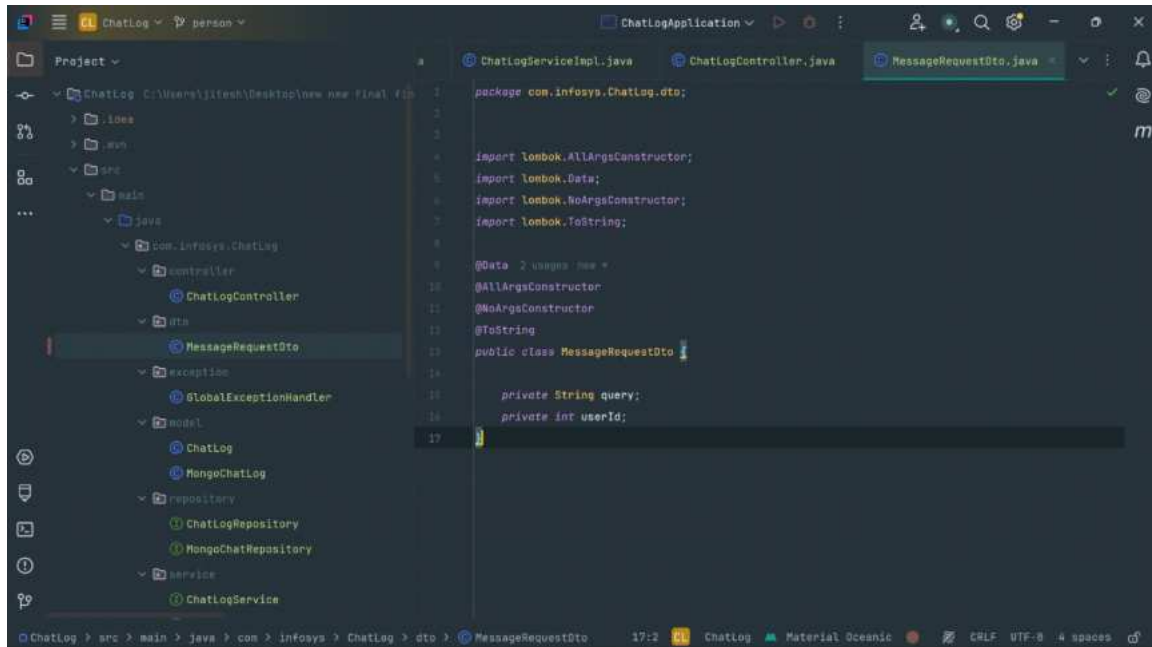


```
1  package com.infosys.ChatLog.model;
2
3  import jakarta.persistence.Entity;
4  import jakarta.persistence.Id;
5  import lombok.*;
6  import org.springframework.data.mongodb.core.mapping.Document;
7
8  import java.time.LocalDateTime;
9
10 @Data
11 @Document(collection = "chatLog")
12 public class MongoChatLog {
13
14     @Id
15     private String id; // Use String type for MongoDB ID
16     private String query;
17     private String response;
18     private LocalDateTime timestamp; // Timestamp to sort by, if needed.
19
20 }
```


DTOs:

Define DTOs to control data sent to the client.

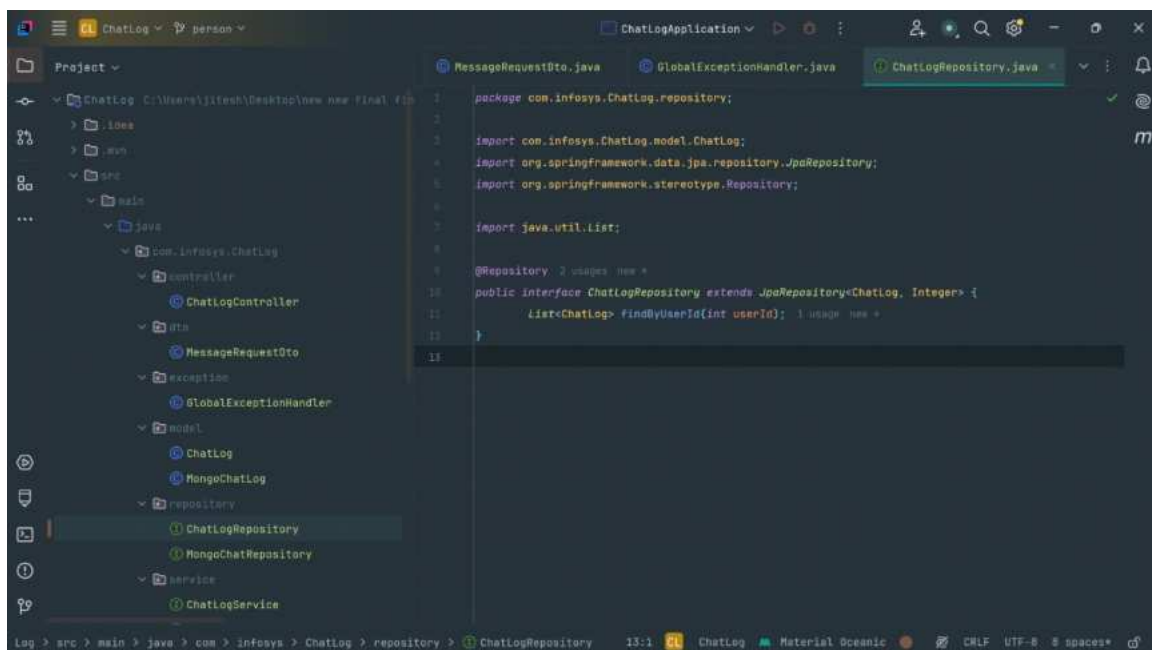
Class: MessageRequestDTO



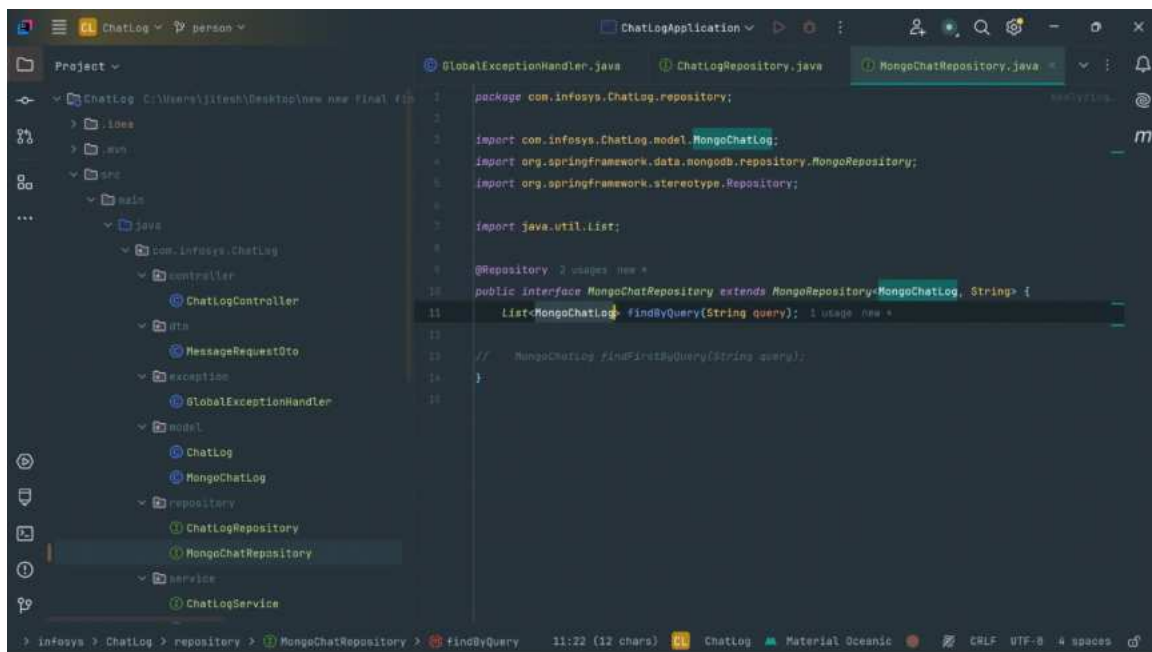
Repository :

Create Repositories to manage entities

Class: ChatLogRepository



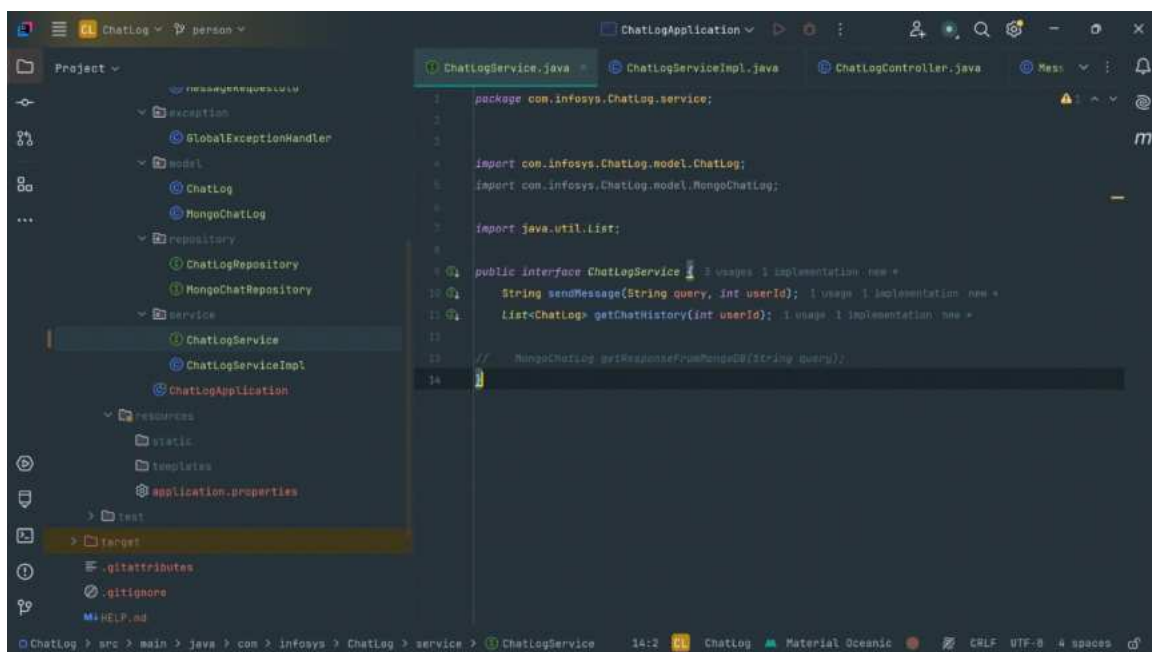
Class: MongoChatLog



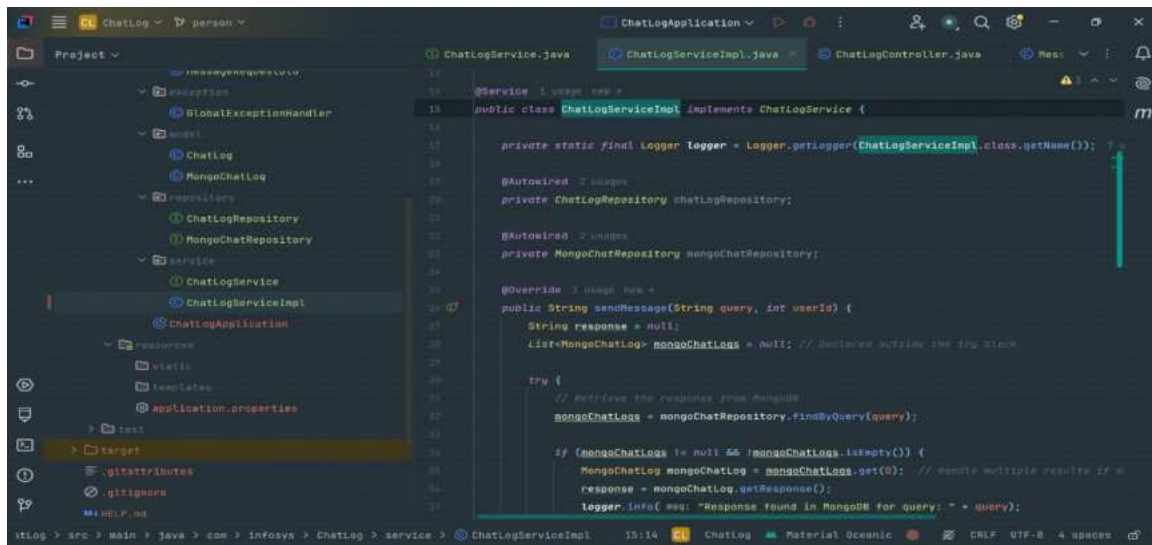
Service:

Create service classes to handle business logic, such as managing conversations and messages.

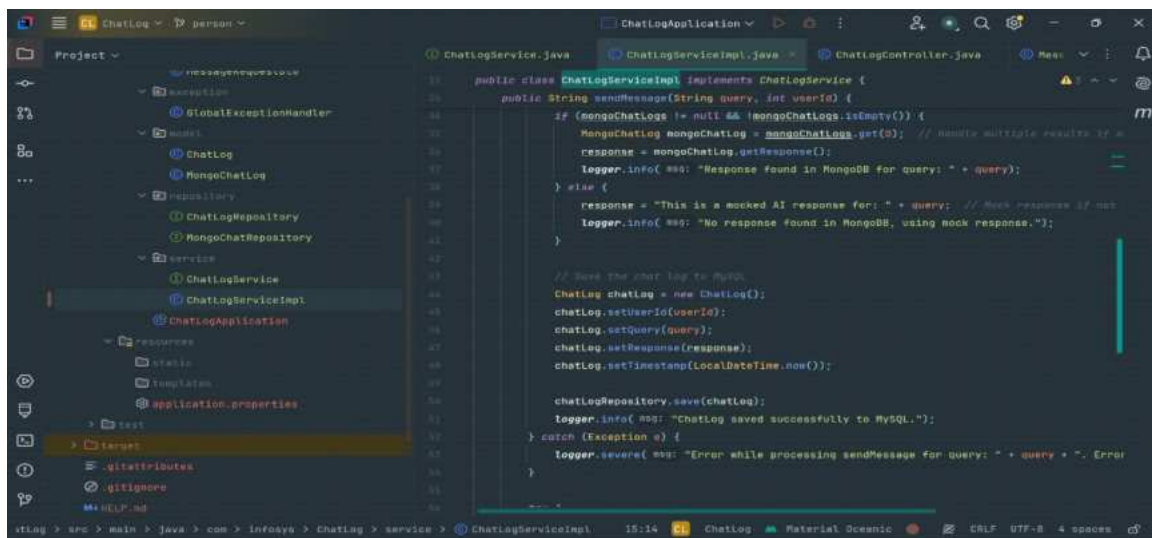
Class: ChatLogService



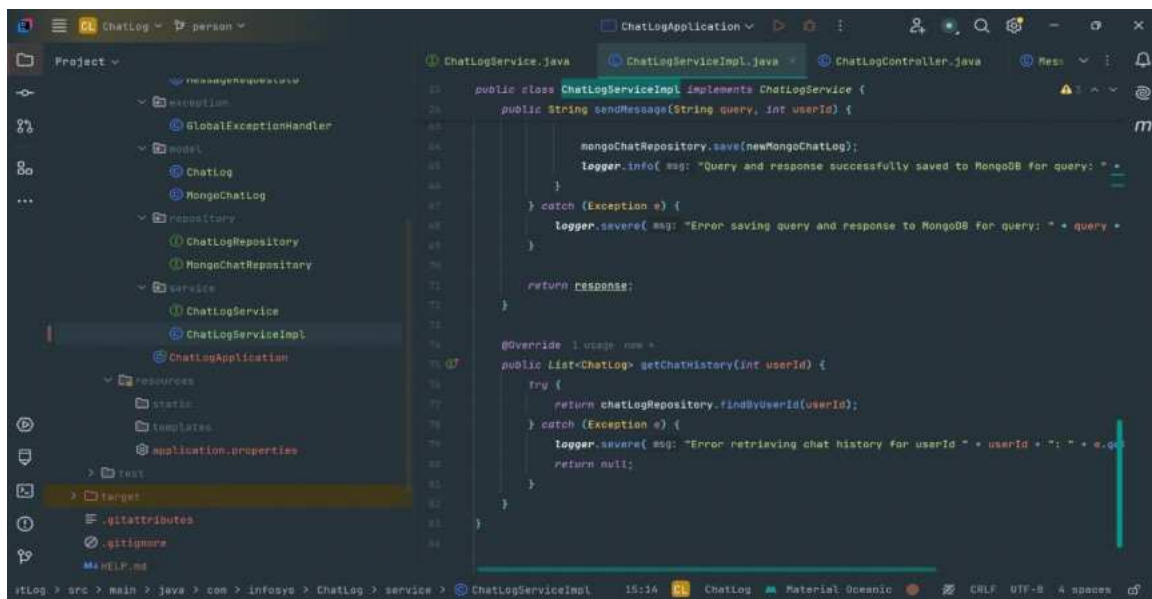
Class: ChatLogServiceImpl



```
11 @Service
12 public class ChatLogServiceImpl implements ChatLogService {
13     private static final Logger logger = Logger.getLogger(ChatLogServiceImpl.class.getName());
14
15     @Autowired
16     private ChatLogRepository chatLogRepository;
17
18     @Autowired
19     private MongoChatRepository mongoChatRepository;
20
21     @Override
22     public String sendMessage(String query, int userId) {
23         String response = null;
24         List<MongoChatLog> mongoChatLogs = null; // declared outside the try block
25
26         try {
27             // Retrieve the response from MongoDB
28             mongoChatLogs = mongoChatRepository.findByQuery(query);
29
30             if (mongoChatLogs != null && !mongoChatLogs.isEmpty()) {
31                 MongoChatLog mongoChatLog = mongoChatLogs.get(0); // handle multiple results if =
32                 response = mongoChatLog.getResponse();
33                 logger.info(msg: "Response found in MongoDB for query: " + query);
34             }
35         } catch (Exception e) {
36             logger.error(msg: "Error while processing sendMessage for query: " + query + ". Error: " + e.getMessage());
37         }
38
39         return response;
40     }
41 }
```



```
11 public class ChatLogServiceImpl implements ChatLogService {
12     public String sendMessage(String query, int userId) {
13         if (mongoChatLogs != null && !mongoChatLogs.isEmpty()) {
14             MongoChatLog mongoChatLog = mongoChatLogs.get(0); // handle multiple results if =
15             response = mongoChatLog.getResponse();
16             logger.info(msg: "Response found in MongoDB for query: " + query);
17         } else {
18             response = "This is a mocked AI response for: " + query; // Mock response if not
19             logger.info(msg: "No response found in MongoDB, using mock response.");
20         }
21
22         // Save the chat log to MySQL
23         ChatLog chatLog = new ChatLog();
24         chatLog.setUserId(userId);
25         chatLog.setQuery(query);
26         chatLog.setResponse(response);
27         chatLog.setTimestamp(LocalDateTime.now());
28
29         chatLogRepository.save(chatLog);
30         logger.info(msg: "ChatLog saved successfully to MySQL.");
31     } catch (Exception e) {
32         logger.error(msg: "Error while processing sendMessage for query: " + query + ". Error: " + e.getMessage());
33     }
34
35     return response;
36 }
37 }
```

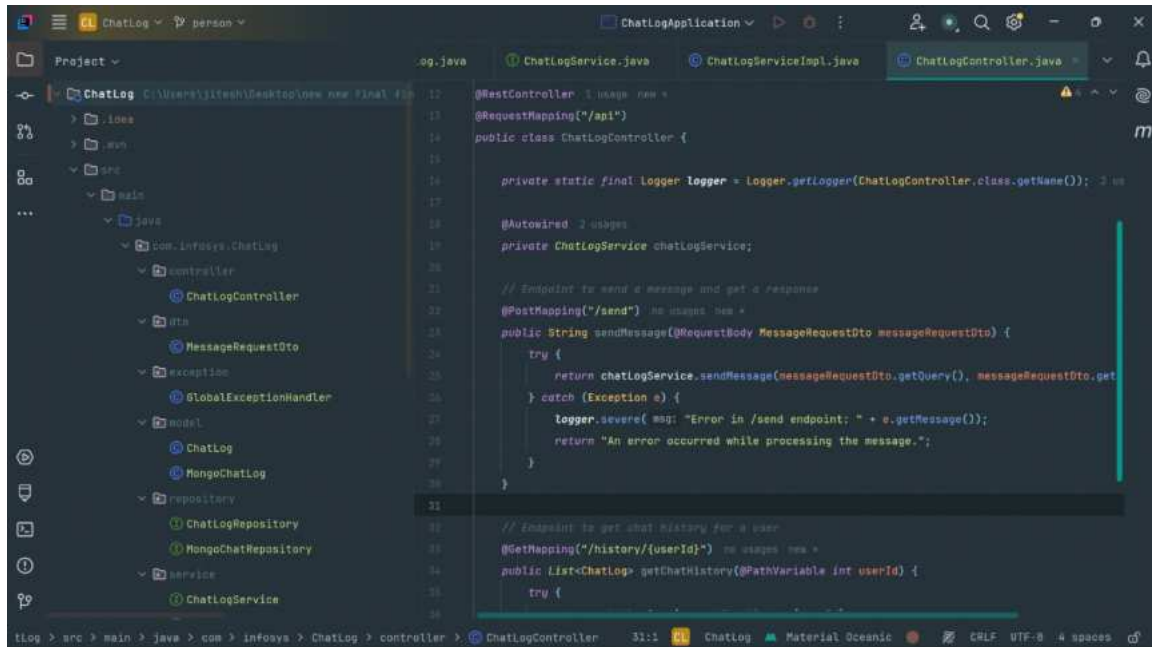


```
11 public class ChatLogServiceImpl implements ChatLogService {
12     public String sendMessage(String query, int userId) {
13         mongoChatRepository.save(new MongoChatLog());
14         logger.info(msg: "Query and response successfully saved to MongoDB for query: " + query);
15     } catch (Exception e) {
16         logger.error(msg: "Error saving query and response to MongoDB for query: " + query + ". Error: " + e.getMessage());
17     }
18
19     return response;
20 }
21
22 @Override
23 public List<ChatLog> getChatHistory(int userId) {
24     try {
25         return chatLogRepository.findByUserId(userId);
26     } catch (Exception e) {
27         logger.error(msg: "Error retrieving chat history for userId: " + userId + ". Error: " + e.getMessage());
28     }
29
30     return null;
31 }
32 }
```

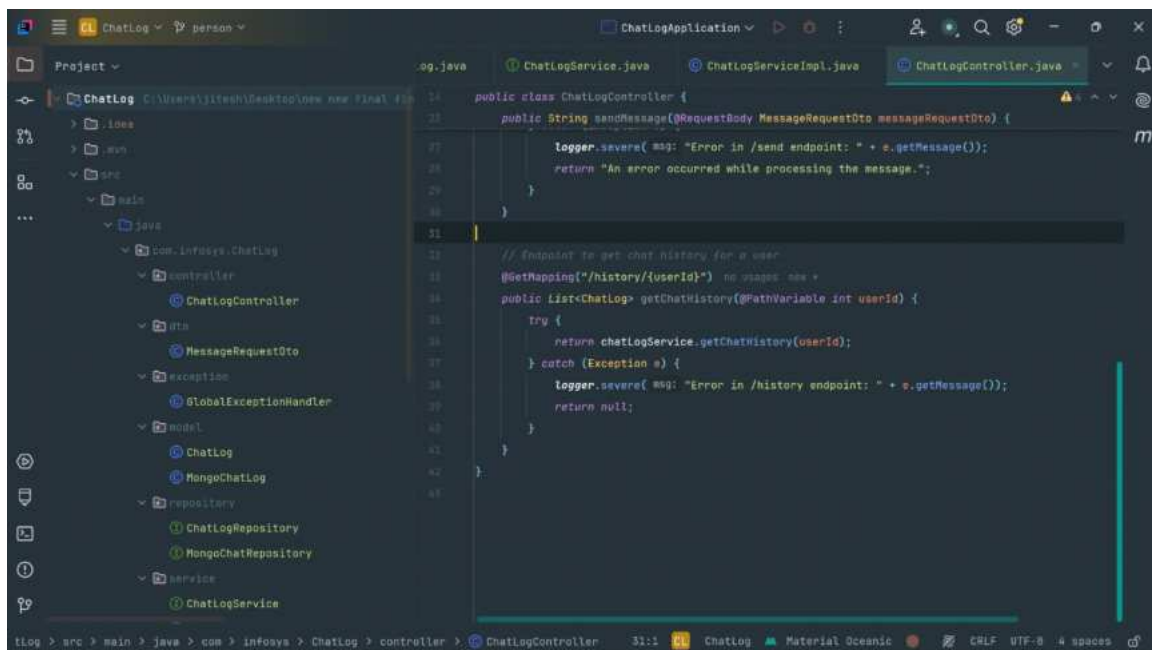
Controller :

Create REST endpoints for Chatbot interaction.

Class: ChatLogController



```
12 @RestController 1 usage new +
13 @RequestMapping("/api")
14 public class ChatLogController {
15
16     private static final Logger logger = Logger.getLogger(ChatLogController.class.getName()); 2 us
17
18     @Autowired 2 usages
19     private ChatLogService chatLogService;
20
21     // Endpoint to send a message and get a response
22     @PostMapping("/send") no usages new +
23     public String sendMessage(@RequestBody MessageRequestDto messageRequestDto) {
24         try {
25             return chatLogService.sendMessage(messageRequestDto.getQuery(), messageRequestDto.get
26         } catch (Exception e) {
27             logger.severe(msg: "Error in /send endpoint: " + e.getMessage());
28             return "An error occurred while processing the message.";
29         }
30     }
31
32     // Endpoint to get chat history for a user
33     @GetMapping("/history/{userId}") no usages new +
34     public List<ChatLog> getChatHistory(@PathVariable int userId) {
35         try {
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```



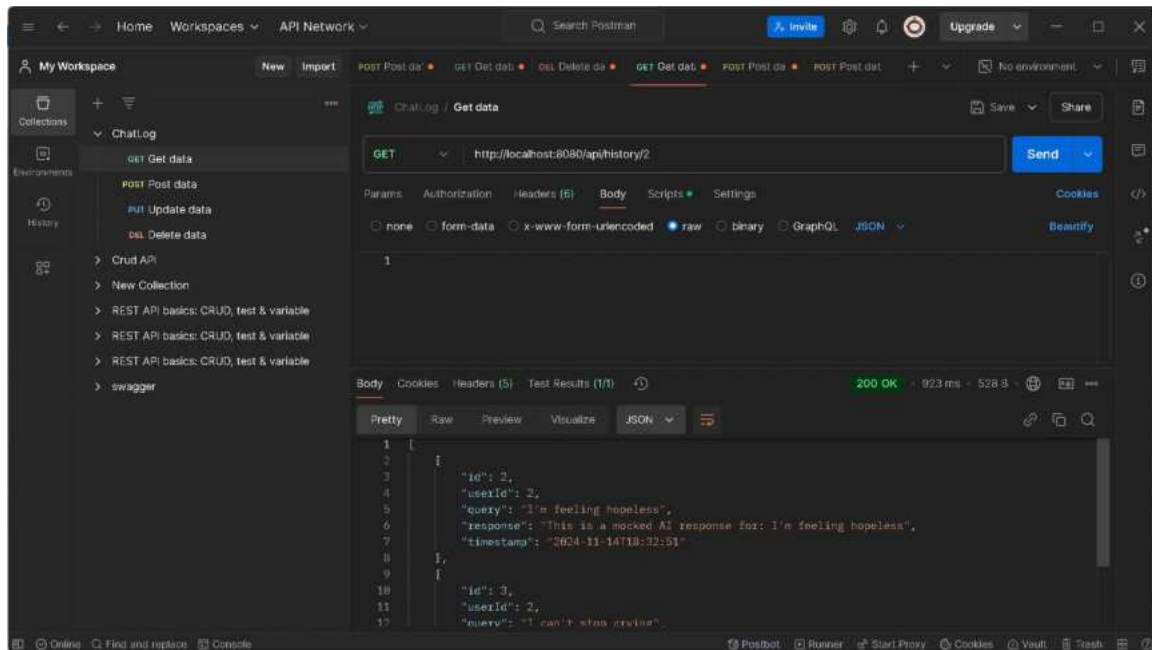
```
14 public class ChatLogController {
15
16     public String sendMessage(@RequestBody MessageRequestDto messageRequestDto) {
17
18         logger.severe(msg: "Error in /send endpoint: " + e.getMessage());
19         return "An error occurred while processing the message.";
20     }
21
22     // Endpoint to get chat history for a user
23     @GetMapping("/history/{userId}") no usages new +
24     public List<ChatLog> getChatHistory(@PathVariable int userId) {
25         try {
26             return chatLogService.getChatHistory(userId);
27         } catch (Exception e) {
28             logger.severe(msg: "Error in /history endpoint: " + e.getMessage());
29             return null;
30         }
31     }
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Testing and Validation

Test the application using Postman or other tools to ensure endpoints work as expected.

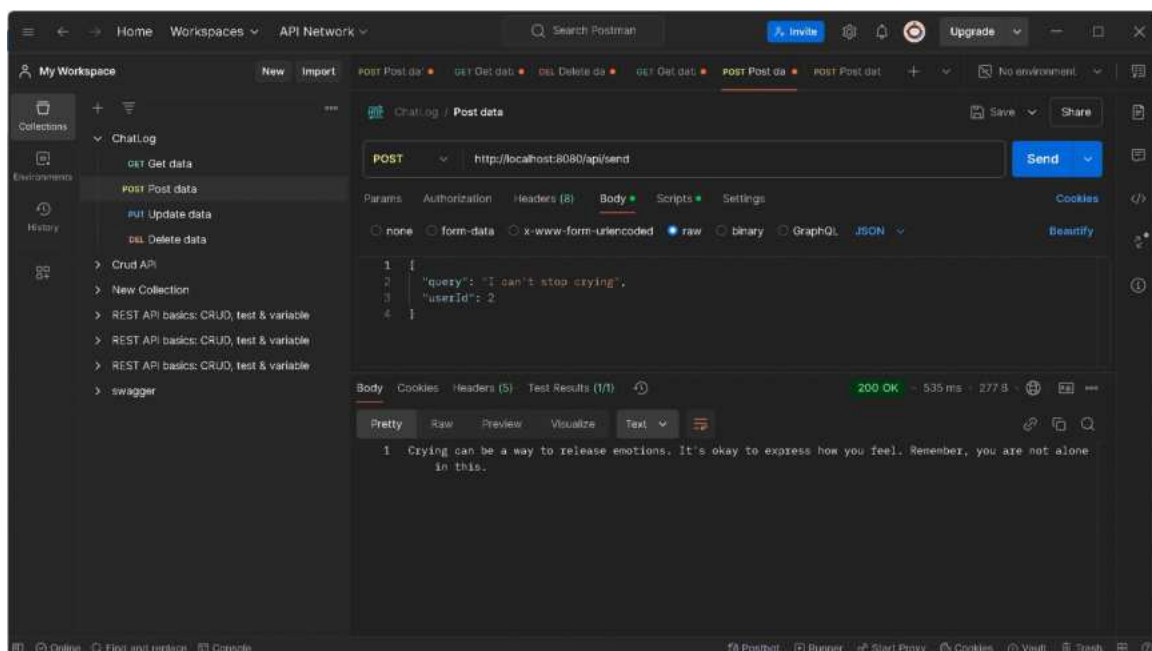
GET

<http://localhost:8080/api/history/2>

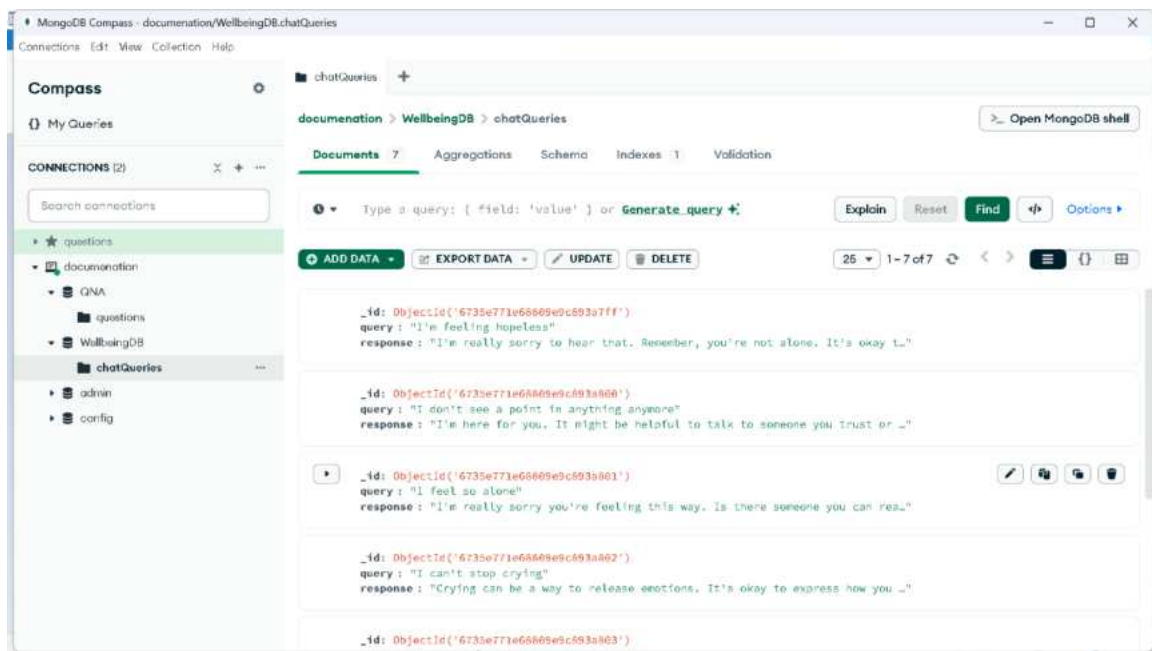


POST

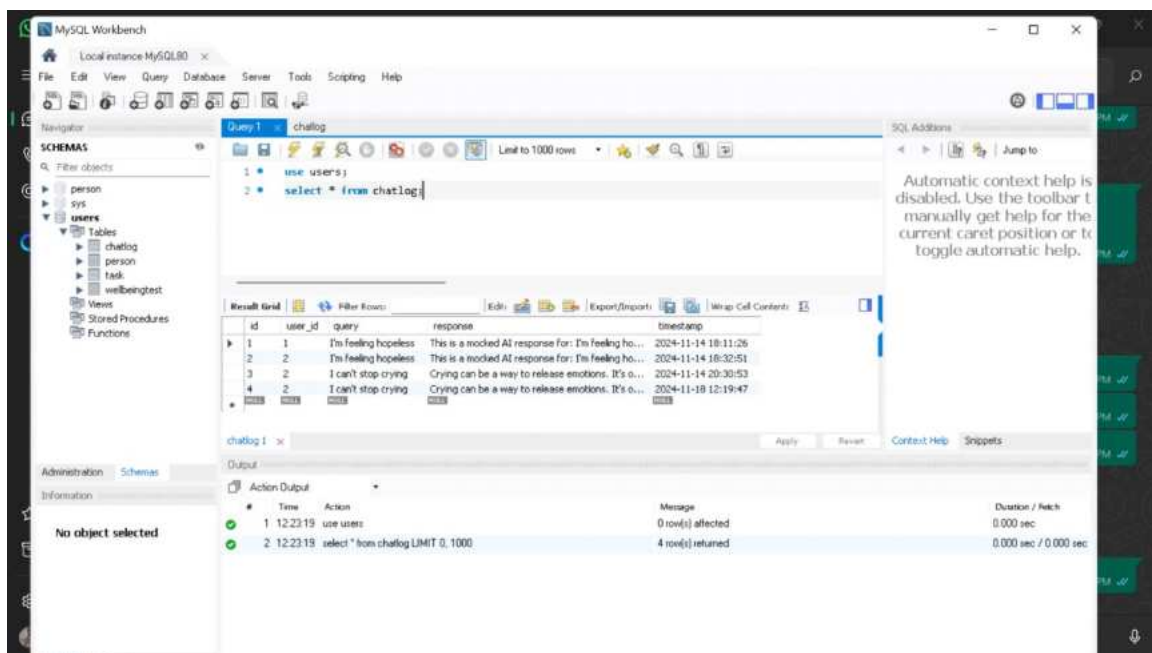
<http://localhost:8080/api/send>



Mongodb



Database for chatlog



Conclusion:

This outline provides a foundational structure for a backend for managing users, conversations, and message handling, with the flexibility to integrate a chatbot AI for responses.

Milestone 4: Task Management

Add Tasks: Enter task details, set a due date, priority, and reminder.

Modify Tasks: Update task details, including title, due date, and reminder settings.

DeleteTasks: Remove tasks that are no longer needed.

Reminders: Receive notifications based on your reminder settings to keep track of tasks.

Dependencies:

→Lombok

→spring data JPA

→Spring web

→MySQL driver

Packages & Classes:

→Controller

- TaskController

→DTO

- TaskDto

→Exception

- GlobalExceptionHandler
- TaskNotFoundException

→Model

- Task

→Repository

- TaskRepo

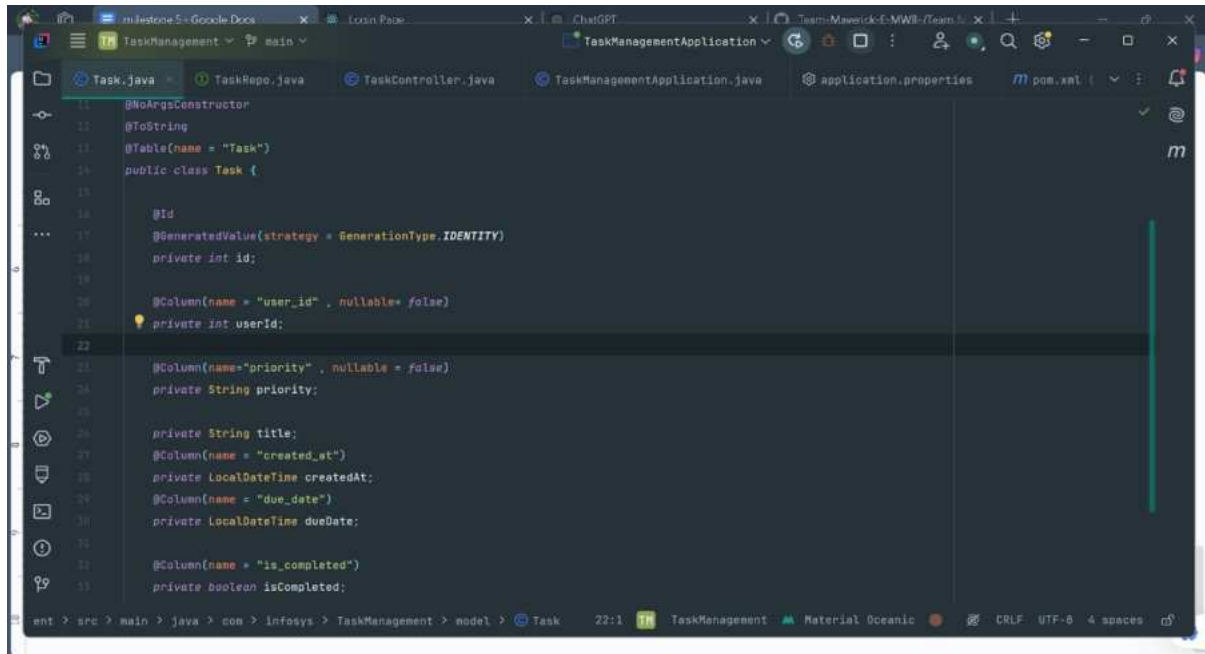
→Service

- TaskService
- TaskServiceImpl

Model:

Represents the data or entity structure that will be stored in the database.

Class: Task

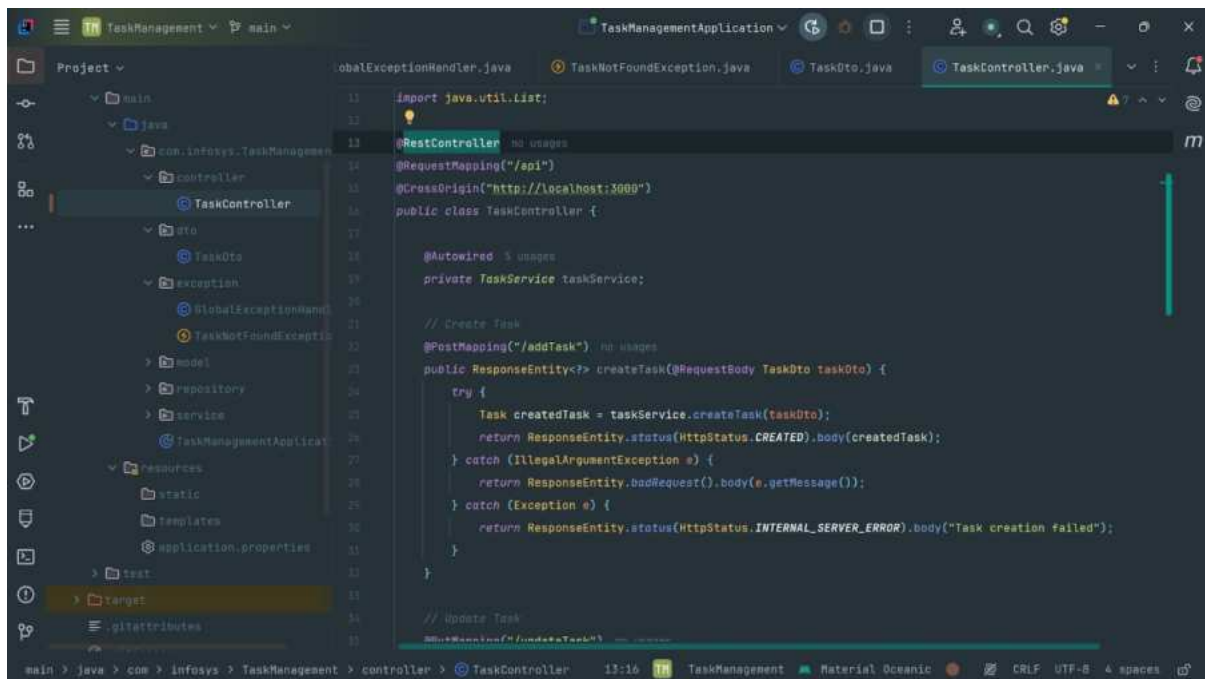


```
11 @NoArgsConstructor
12 @ToString
13 @Table(name = "Task")
14 public class Task {
15
16     @Id
17     @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private int id;
19
20     @Column(name = "user_id", nullable = false)
21     private int userId;
22
23     @Column(name = "priority", nullable = false)
24     private String priority;
25
26     private String title;
27     @Column(name = "created_at")
28     private LocalDateTime createdAt;
29     @Column(name = "due_date")
30     private LocalDateTime dueDate;
31
32     @Column(name = "is_completed")
33     private boolean isCompleted;
```

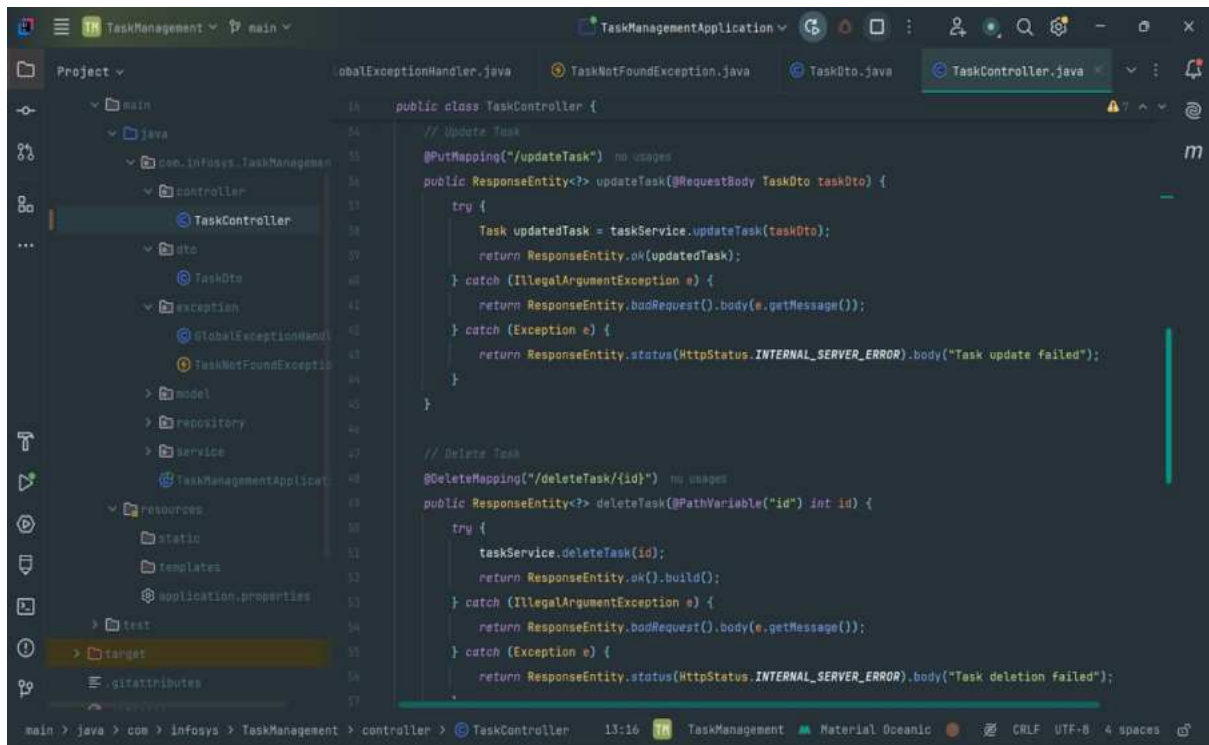
Controller:

Acts as the entry point for client requests (typically HTTP requests in a REST API).

Class: TaskController

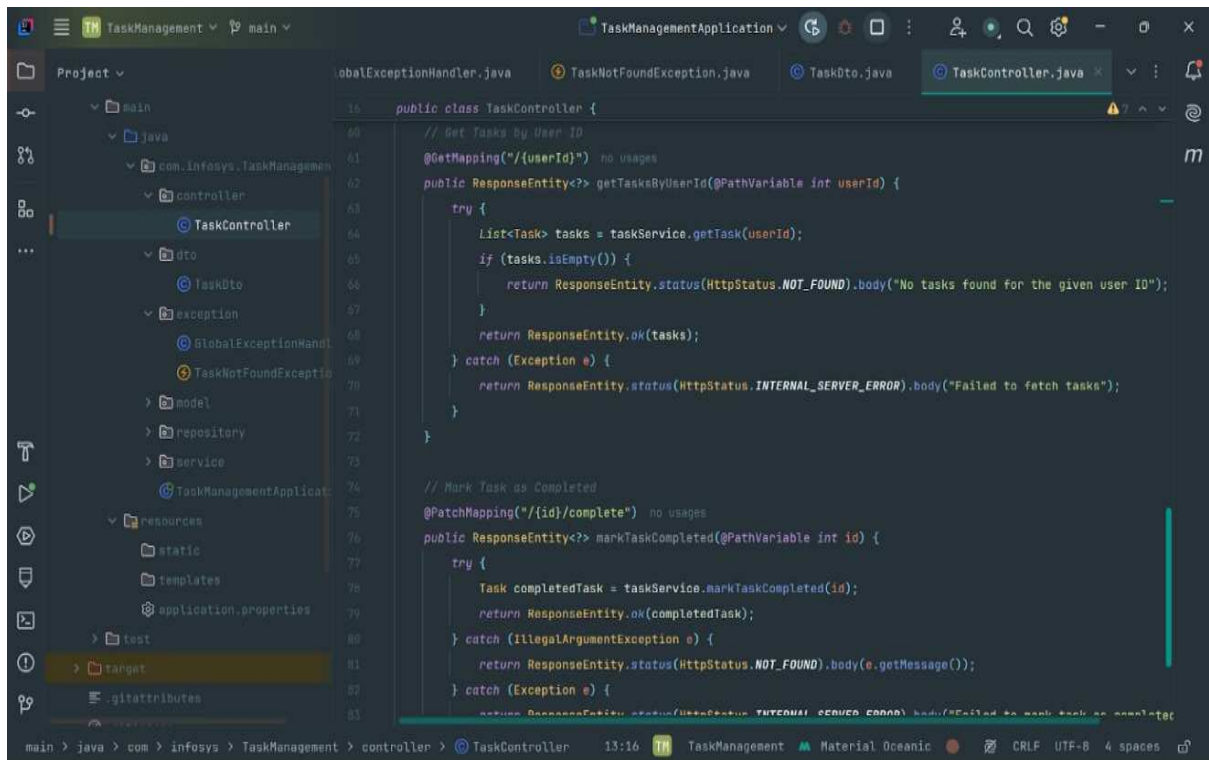


```
11 import java.util.List;
12
13 @RestController
14 @RequestMapping("/api")
15 @CrossOrigin("http://localhost:3000")
16 public class TaskController {
17
18     @Autowired
19     private TaskService taskService;
20
21     // Create Task
22     @PostMapping("/addTask")
23     public ResponseEntity<?> createTask(@RequestBody TaskDto taskDto) {
24         try {
25             Task createdTask = taskService.createTask(taskDto);
26             return ResponseEntity.status(HttpStatus.CREATED).body(createdTask);
27         } catch (IllegalArgumentException e) {
28             return ResponseEntity.badRequest().body(e.getMessage());
29         } catch (Exception e) {
30             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Task creation failed");
31         }
32     }
33
34     // Update Task
35     @PutMapping("/updateTask/{id}")
```



The screenshot shows an IDE window for a project named 'TaskManagement'. The 'TaskController.java' file is open, displaying two REST API endpoints. The first endpoint, `@PostMapping("/updateTask")`, handles updating a task by its ID. It uses `taskService.updateTask(taskDto)` and returns the updated task. The second endpoint, `@DeleteMapping("/deleteTask/{id}")`, handles deleting a task by its ID. It uses `taskService.deleteTask(id)` and returns a success response. Both methods include exception handling for `IllegalArgumentException` and a general `Exception`.

```
14 public class TaskController {
15     // Update Task
16     @PostMapping("/updateTask") no usages
17     public ResponseEntity<?> updateTask(@RequestBody TaskDto taskDto) {
18         try {
19             Task updatedTask = taskService.updateTask(taskDto);
20             return ResponseEntity.ok(updatedTask);
21         } catch (IllegalArgumentException e) {
22             return ResponseEntity.badRequest().body(e.getMessage());
23         } catch (Exception e) {
24             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Task update failed");
25         }
26     }
27
28     // Delete Task
29     @DeleteMapping("/deleteTask/{id}") no usages
30     public ResponseEntity<?> deleteTask(@PathVariable("id") int id) {
31         try {
32             taskService.deleteTask(id);
33             return ResponseEntity.ok().build();
34         } catch (IllegalArgumentException e) {
35             return ResponseEntity.badRequest().body(e.getMessage());
36         } catch (Exception e) {
37             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Task deletion failed");
38         }
39     }
40 }
```



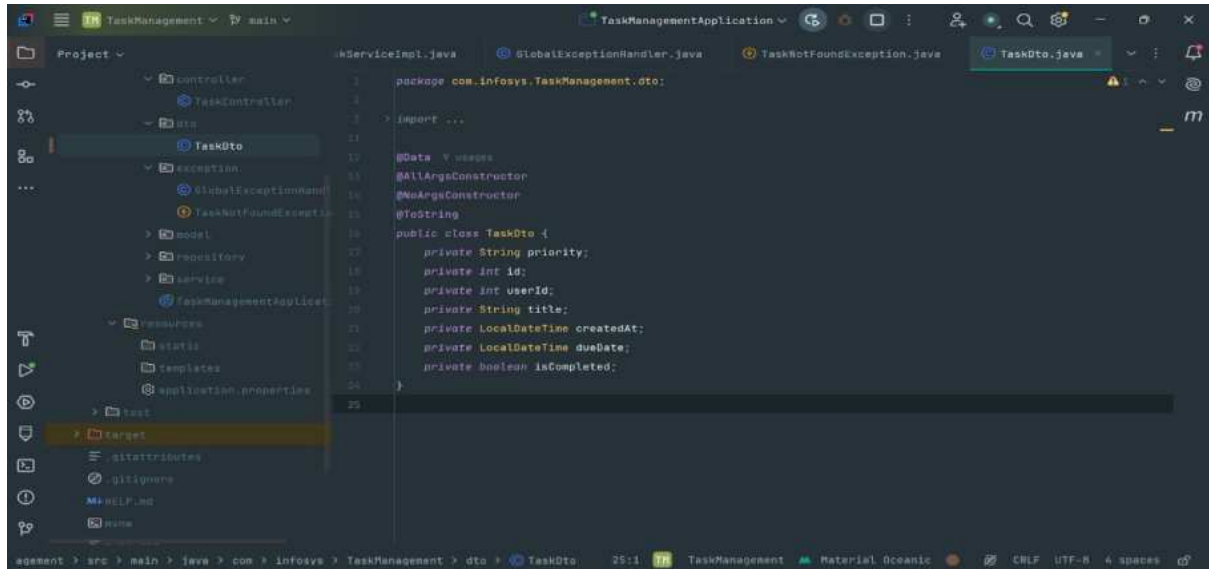
The screenshot shows the same IDE window, but now displaying two different REST API endpoints in the `TaskController.java` file. The first endpoint, `@GetMapping("/{userId}")`, handles retrieving tasks for a specific user. It uses `taskService.getTask(userId)` and returns the tasks if they exist, or a `NOT_FOUND` response if not. The second endpoint, `@PatchMapping("/{id}/complete")`, handles marking a task as completed. It uses `taskService.markTaskCompleted(id)` and returns the completed task. Both methods include exception handling for `IllegalArgumentException` and a general `Exception`.

```
60 public class TaskController {
61     // Get Tasks by User ID
62     @GetMapping("/{userId}") no usages
63     public ResponseEntity<?> getTasksByUserId(@PathVariable int userId) {
64         try {
65             List<Task> tasks = taskService.getTask(userId);
66             if (tasks.isEmpty()) {
67                 return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No tasks found for the given user ID");
68             }
69             return ResponseEntity.ok(tasks);
70         } catch (Exception e) {
71             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to fetch tasks");
72         }
73     }
74
75     // Mark Task as Completed
76     @PatchMapping("/{id}/complete") no usages
77     public ResponseEntity<?> markTaskCompleted(@PathVariable int id) {
78         try {
79             Task completedTask = taskService.markTaskCompleted(id);
80             return ResponseEntity.ok(completedTask);
81         } catch (IllegalArgumentException e) {
82             return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
83         } catch (Exception e) {
84             return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to mark task as completed");
85         }
86     }
87 }
```

Dto:

Transfers data between different layers (especially between Controller and Service) without exposing the entity directly.

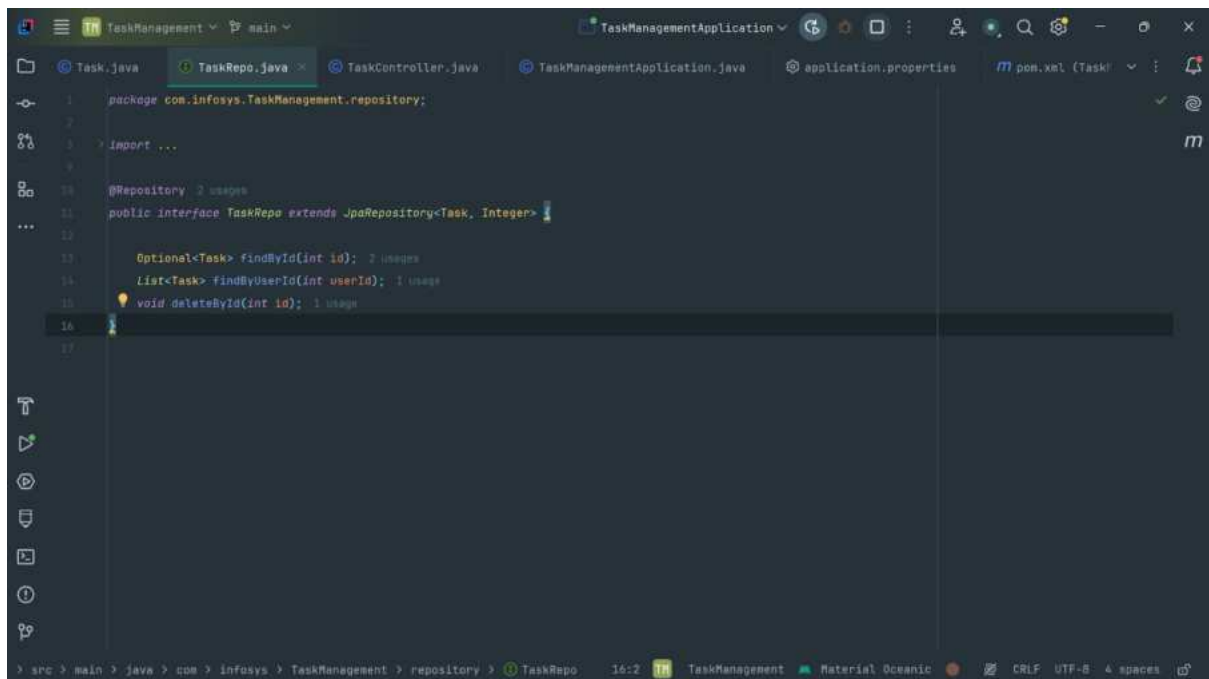
Class: TaskDto



Repository

Handles the interaction with the database using JPA (Java Persistence API).

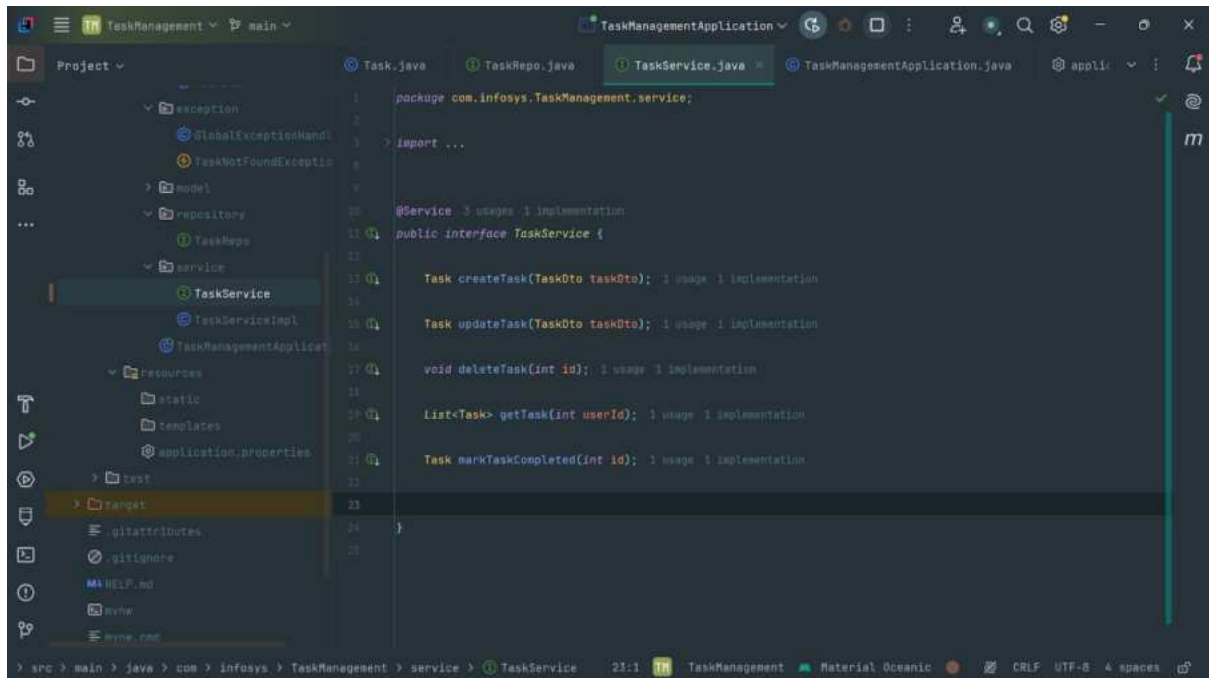
Class: TaskRepo



Service:

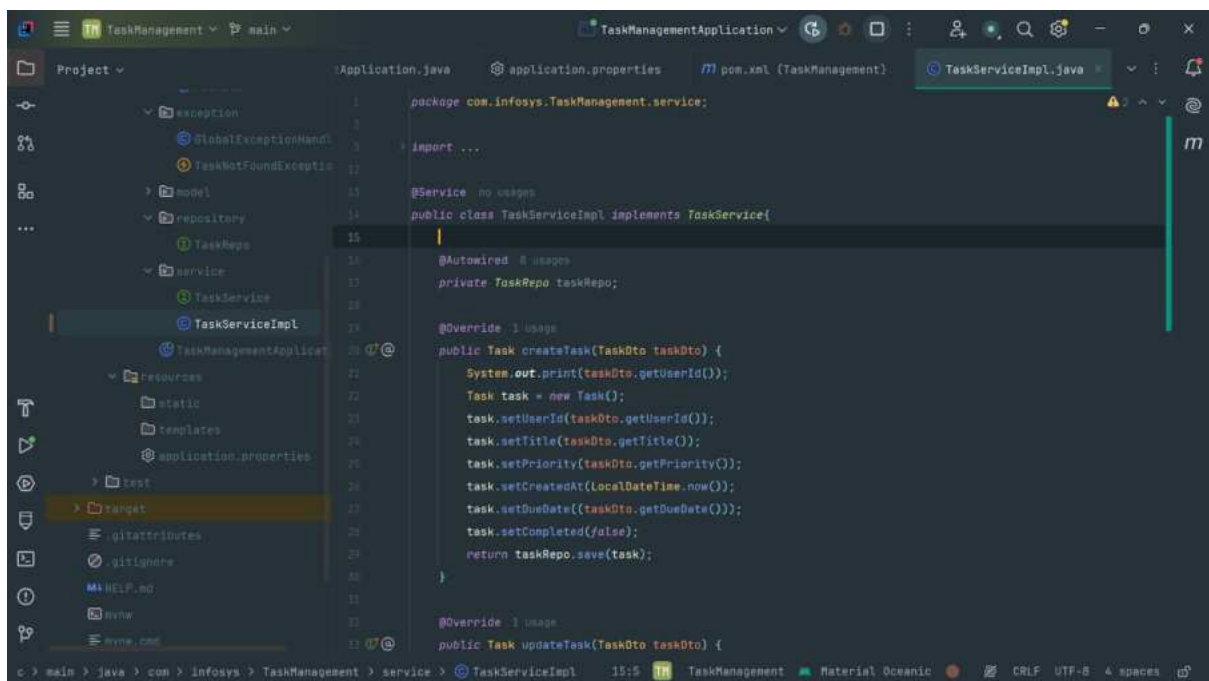
Contains the business logic. It processes the data and communicates between the Controller and Repository layers.

Class: TaskService

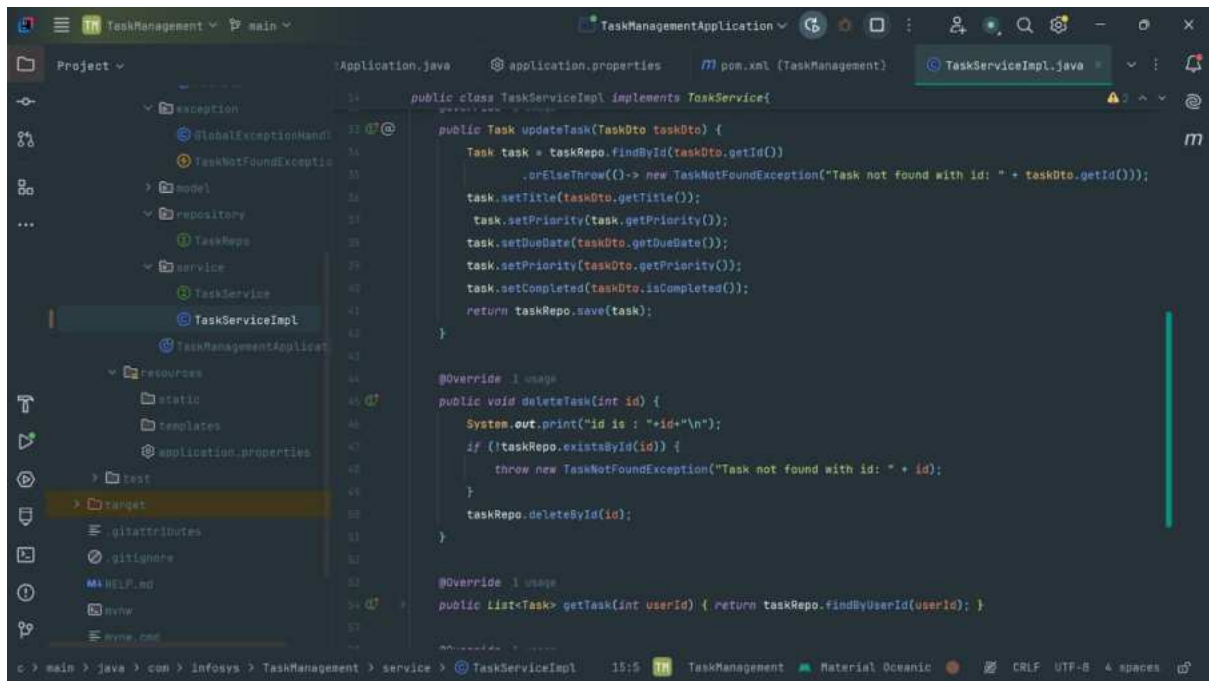


```
1 package com.infosys.TaskManagement.service;
2
3 import ...
4
5 @Service // usage: 1 implementation
6 public interface TaskService {
7
8     Task createTask(TaskDto taskDto); // usage: 1 implementation
9
10    Task updateTask(TaskDto taskDto); // usage: 1 implementation
11
12    void deleteTask(int id); // usage: 1 implementation
13
14    List<Task> getTask(int userId); // usage: 1 implementation
15
16    Task markTaskCompleted(int id); // usage: 1 implementation
17
18 }
```

Class: TaskServiceImpl



```
1 package com.infosys.TaskManagement.service;
2
3 import ...
4
5 @Service // no usages
6 public class TaskServiceImpl implements TaskService {
7
8     @Autowired // usage: 1
9     private TaskRepo taskRepo;
10
11     @Override // usage: 1
12     public Task createTask(TaskDto taskDto) {
13         System.out.println(taskDto.getUserId());
14         Task task = new Task();
15         task.setUserId(taskDto.getUserId());
16         task.setTitle(taskDto.getTitle());
17         task.setPriority(taskDto.getPriority());
18         task.setCreatedAt(LocalDate.now());
19         task.setDueDate(taskDto.getDueDate());
20         task.setCompleted(false);
21         return taskRepo.save(task);
22     }
23
24     @Override // usage: 1
25     public Task updateTask(TaskDto taskDto) {
26
27     }
```



```
public class TaskServiceImpl implements TaskService {

    public Task updateTask(TaskDto taskDto) {
        Task task = taskRepo.findById(taskDto.getId())
            .orElseThrow(() -> new TaskNotFoundException("Task not found with id: " + taskDto.getId()));
        task.setTitle(taskDto.getTitle());
        task.setPriority(taskDto.getPriority());
        task.setDueDate(taskDto.getDueDate());
        task.setPriority(taskDto.getPriority());
        task.setCompleted(taskDto.isCompleted());
        return taskRepo.save(task);
    }

    @Override // usage
    public void deleteTask(int id) {
        System.out.println("id is: " + id + "\n");
        if (!taskRepo.existsById(id)) {
            throw new TaskNotFoundException("Task not found with id: " + id);
        }
        taskRepo.deleteById(id);
    }

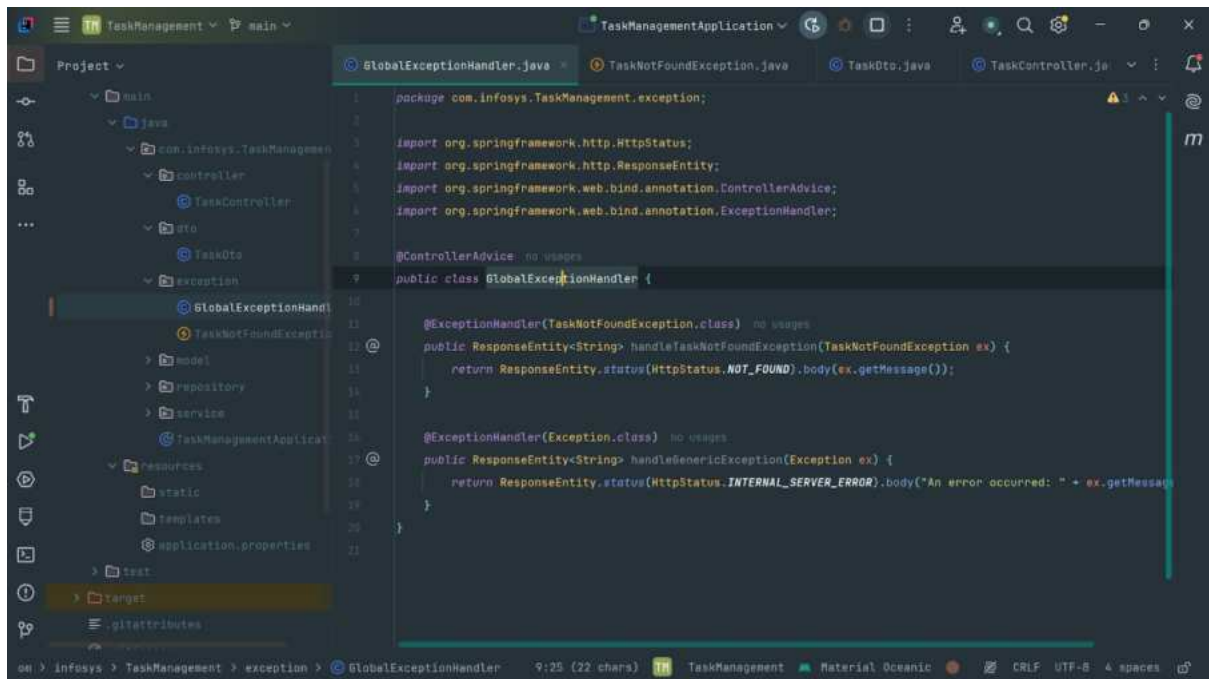
    @Override // usage
    public List<Task> getTask(int userId) { return taskRepo.findById(userId); }

}
```

Exception:

Defines custom error scenarios.

Class: GlobalExceptionHandler



```
package com.infosys.TaskManagement.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;

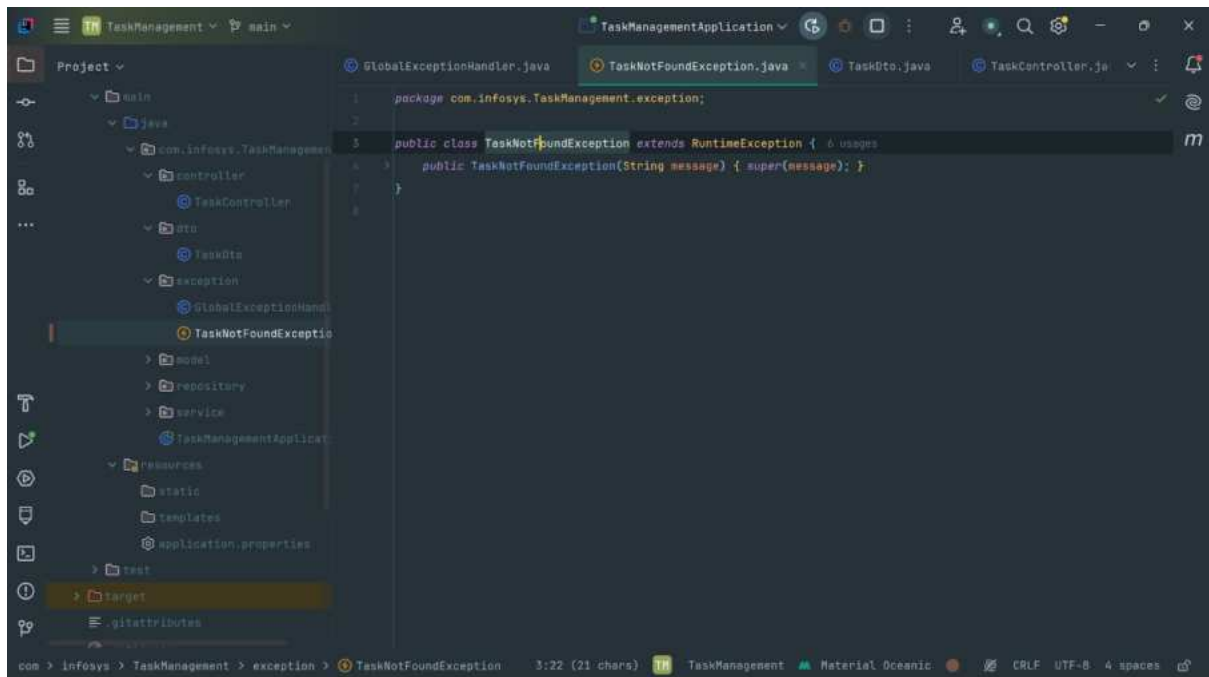
@ControllerAdvice // no usages
public class GlobalExceptionHandler {

    @ExceptionHandler(TaskNotFoundException.class) // no usages
    public ResponseEntity<String> handleTaskNotFoundException(TaskNotFoundException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }

    @ExceptionHandler(Exception.class) // no usages
    public ResponseEntity<String> handleGenericException(Exception ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("An error occurred: " + ex.getMessage());
    }

}
```

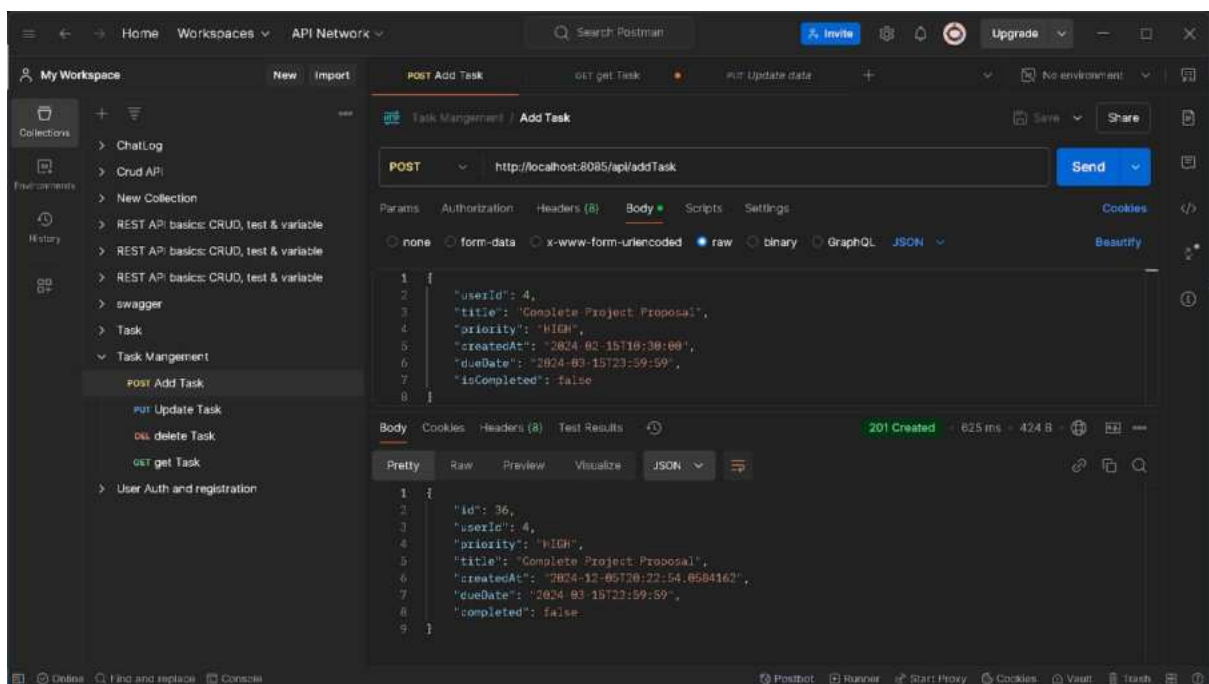
Class: TaskNotFoundException



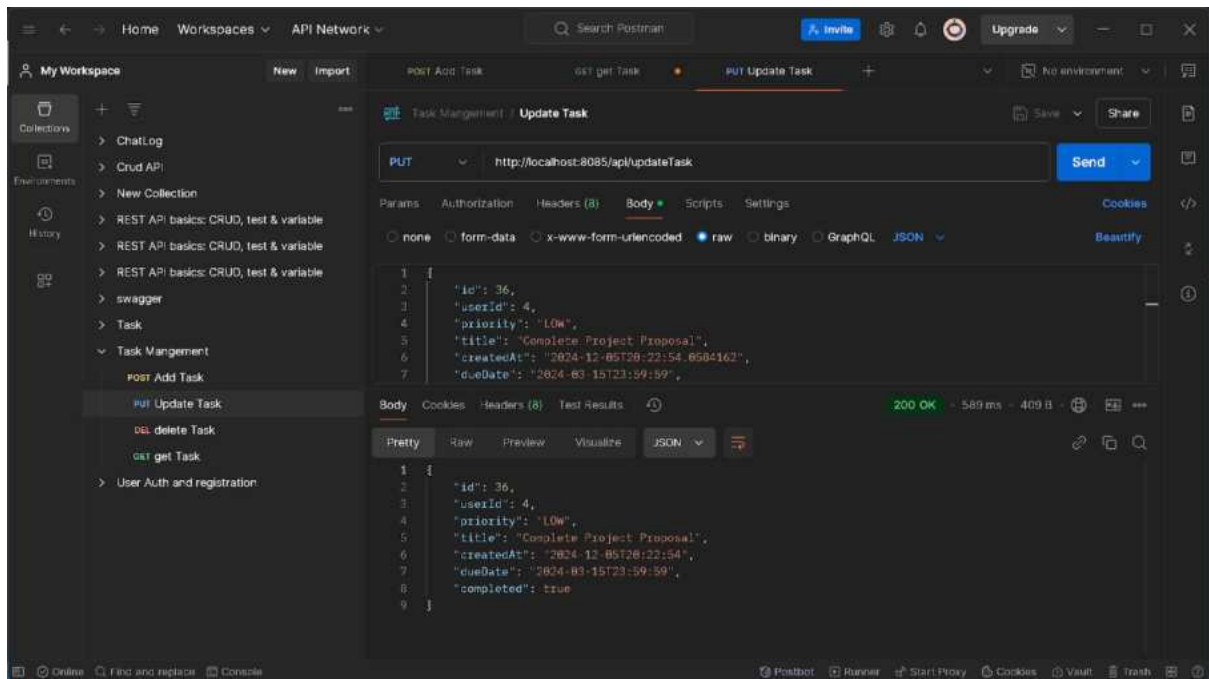
Testing and Validation

Test the application using Postman or other tools to ensure endpoints work as expected.

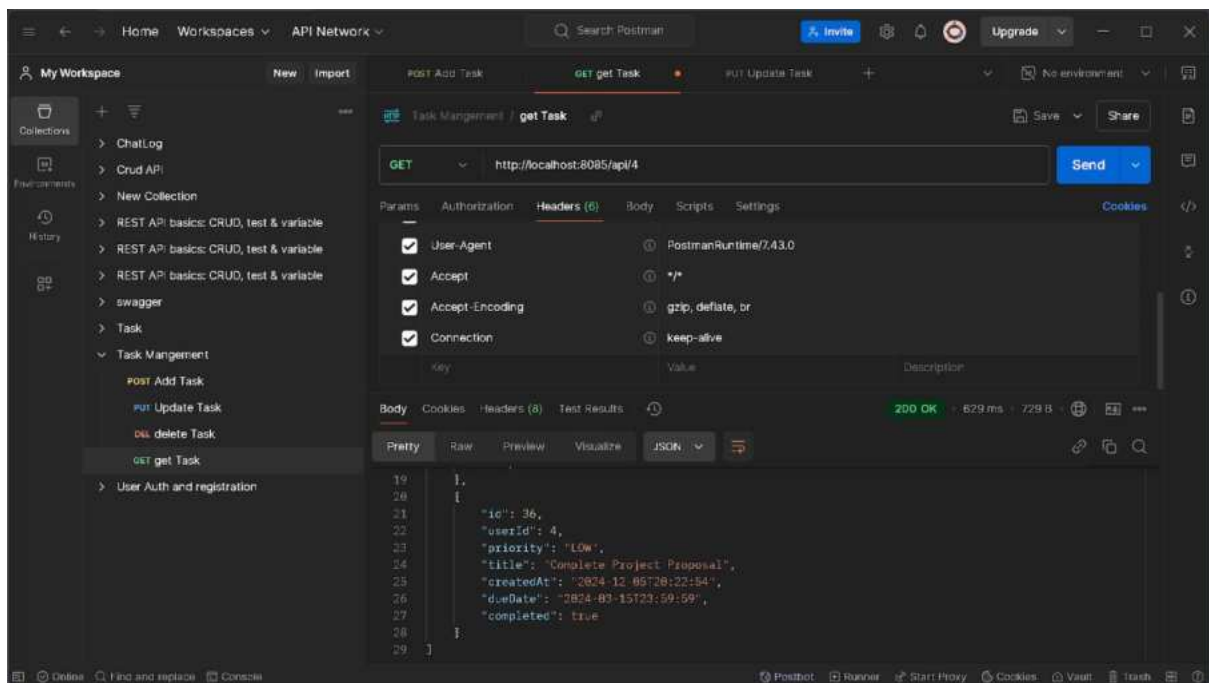
POST: api/addTask



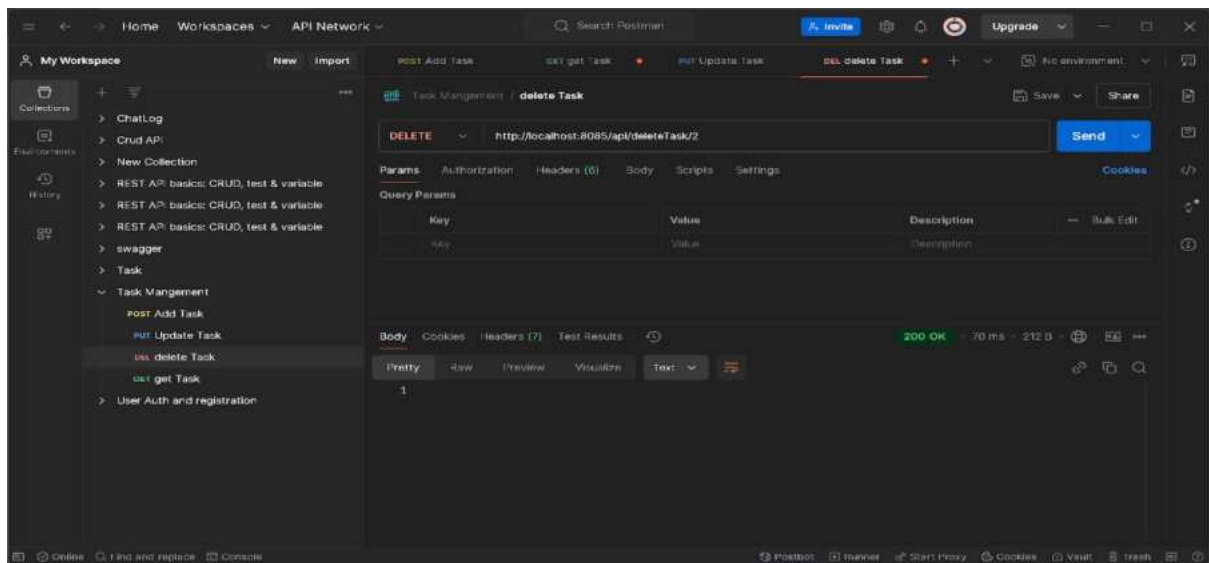
PUT: api/updateTask



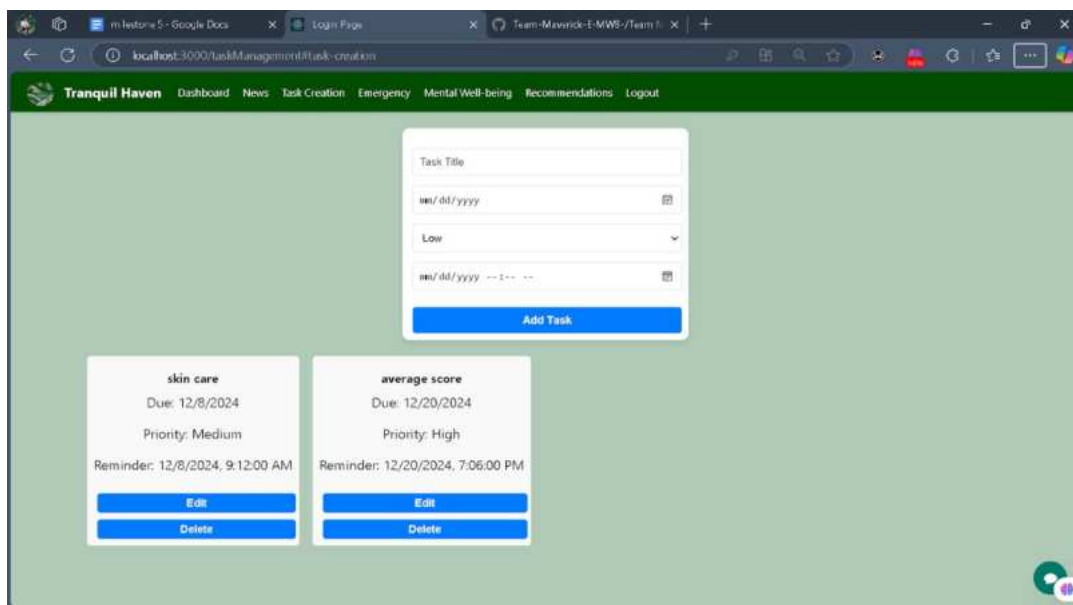
GET : api/{userId}



DELETE : api/deleteTask/{id}



Output



Conclusion:

This Outline provides a foundational structure for a backend for Task Management.