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

→Serive

- PersonService

Model :

Define models to represent user data.

Class: Person

```java
@Data
@ToString
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "person")
public class Person {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "first_name")
    private String firstname;

    @Column(name = "last_name")
    private String lastname;

    private String username;

    private String email;

    @Column(name = "password_hash")
    private String password;
```

Controller:

Define Controller to HTTP requests.

Class: AuthController

```java
@RestController
@RequestMapping("/api")
@CrossOrigin("http://localhost:3000")
public class AuthController {

    @Autowired
    private PersonService personService;

    @Autowired
    private PersonRepository personRepository;

    // Get person by ID
    @GetMapping("/{id}")
    public ResponseEntity<Object> person(@PathVariable int id, Authentication auth) {
        var response = new HashMap<String, Object>();

        Optional<Person> personOptional = personRepository.findById(id);
        if (personOptional.isPresent()) {
            Person person = personOptional.get();
            response.put("Id", person.getId());  // Assuming your Person has a getId() method
            response.put("Authorities", auth.getAuthorities());
            response.put("Person", person);
            return ResponseEntity.ok(response);
        } else {
```

```java
public class AuthController {
    }

    @PostMapping("/register")
    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")
    public ResponseEntity<Object> login(@Valid @RequestBody LoginDto loginDto, BindingResult result) {
```

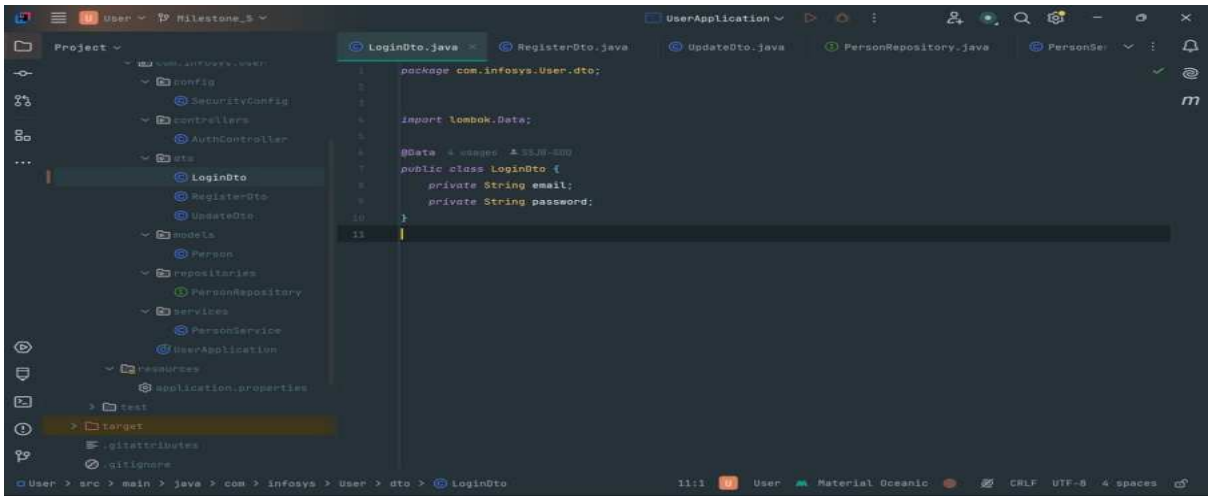
```java
    @PostMapping("/login")
    public ResponseEntity<Object> login(@Valid @RequestBody LoginDto loginDto, BindingResult result) {

        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.print("Hello");
            return ResponseEntity.badRequest().body(response);
        }
        Optional<Person> person = personRepository.findByEmail(loginDto.getEmail());
        response.put("id",person.get().getId());
        return ResponseEntity.ok(response);
    }


    @PutMapping("/update/{username}")
```

```java
    }


    @PutMapping("/update/{username}")
    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);
    }

}
```

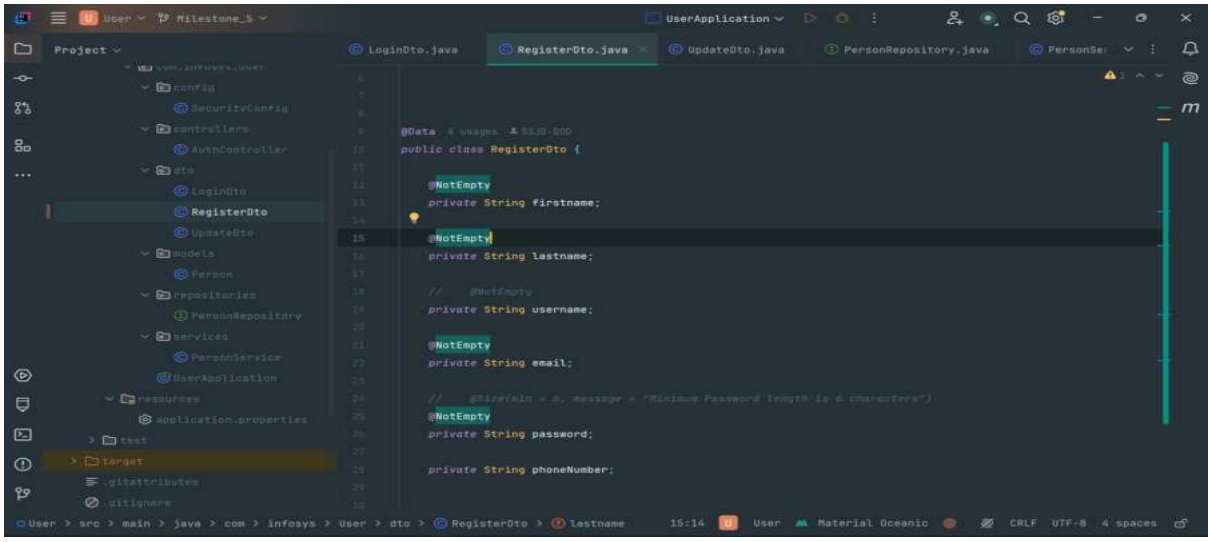
DTOs:

Define DTOs to handle registration data.

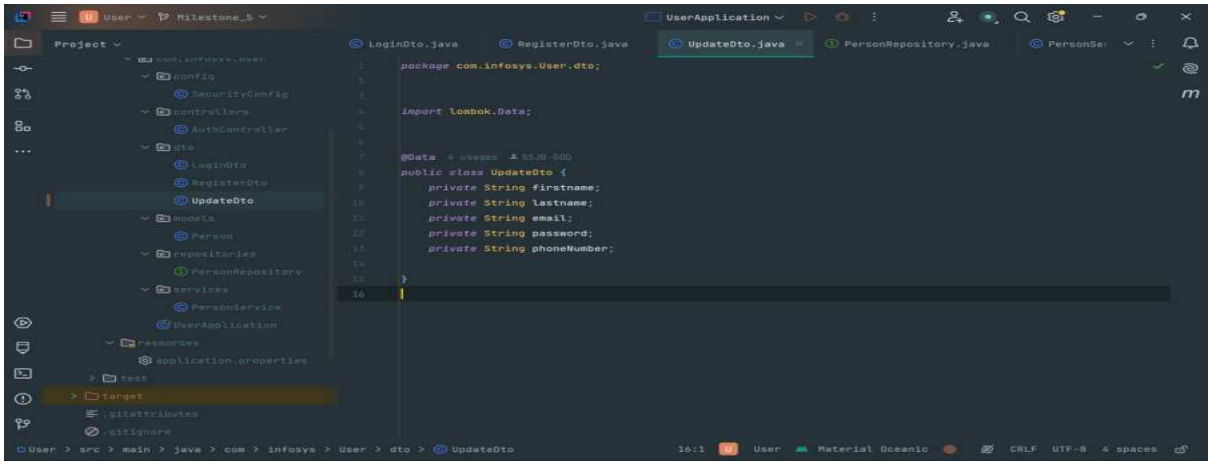
Class: LoginDto



Class: RegisterDto



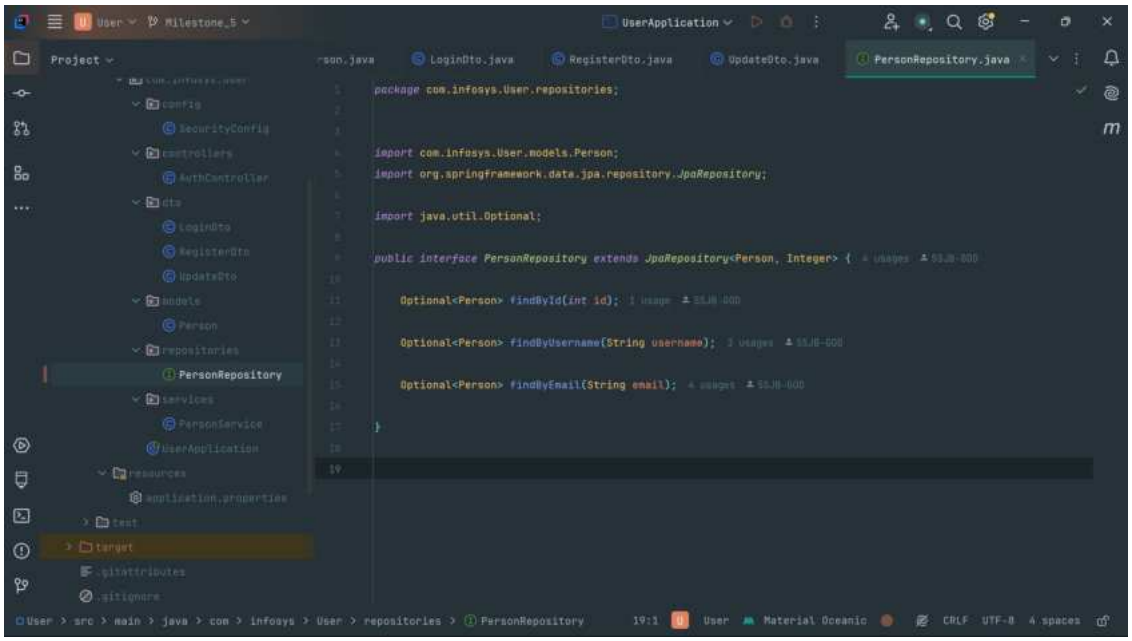
Class: UpdateDto



Repository :

Create Repositories for database interactions

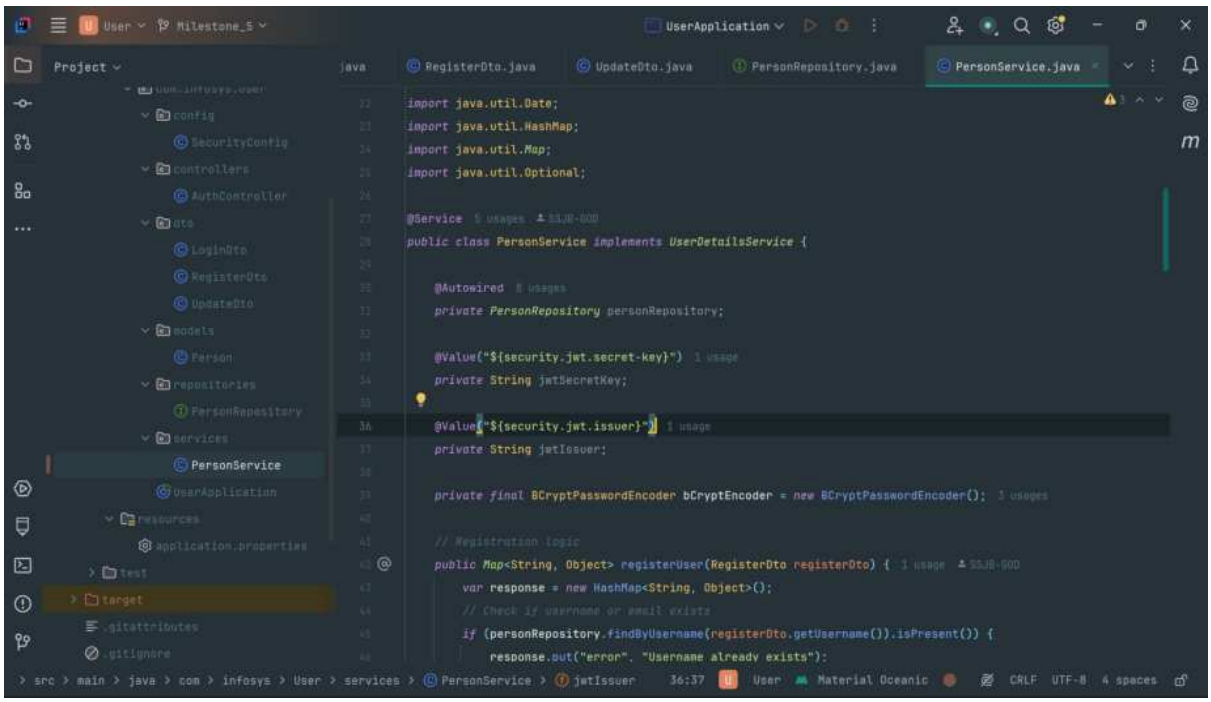
Class: PersonRepository



Service:

Create service classes to handle business logic.

Class: PersonService



```java
public class PersonService implements UserDetailsService {
    public Map<String, Object> registerUser(RegisterDto registerDto) {
        // Check if username or email exists
        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(bCryptEncoder.encode(registerDto.getPassword()));
        personRepository.save(person);

        response.put("token", createJwtToken(person));
        response.put("user", person);
        return response;
    }
```

```java
public class PersonService implements UserDetailsService {
    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 (bCryptEncoder.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");
```

```java
    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;
        }

        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;
            }
```

```java
public class PersonService implements UserDetailsService {
    // JWT creation logic
    private String createJwtToken(Person person) {
        Instant now = Instant.now();
        JwtClaimsSet claims = JwtClaimsSet.builder()
                .issuer(jwtIssuer)
                .issuedAt(now)
                .expiresAt(now.plusSeconds( secondsToAdd: 24 * 3600)) // 1 day expiration
                .subject(person.getUsername())
                .build();

        var encoder = new NimbusJwtEncoder(new ImmutableSecret<>(jwtSecretKey.getBytes()));
        var params = JwtEncoderParameters.from(JwsHeader.with(MacAlgorithm.HS256).build(), claims);
        return encoder.encode(params).getTokenValue();
    }

    // loadUserByUsernme (existing method)
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        return personRepository.findByUsernme(username) Optional<Person>
                .map(person -> User.withUsernme(person.getUsernme())
                        .password(person.getPassword())
                        .build()) Optional<UserDetails>
                .orElseThrow(() -> new UsernameNotFoundException("User not found"));
    }
```

Config:

Create Config for Security Configurations.

Class:SecurityConfig

```java
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Value("${security.jwt.secret-key}")
    private String jwtSecretKey;

    @Autowired
    private PersonService personService;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception{
        return http
                .csrf(csrf -> csrf.disable())
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers( ...patterns: "/").permitAll()
                        .requestMatchers( ...patterns: "/api/login").permitAll()
                        .requestMatchers( ...patterns: "/api/register").permitAll()
                        .anyRequest().permitAll() //(2)
                )
                .oauth2ResourceServer(oauth2 -> oauth2.jwt(Customizer.withDefaults()))
                .sessionManagement(session -> session.sessionCreationPolicy(
                        SessionCreationPolicy.STATELESS))
                .build();
```

```java
public class SecurityConfig {


    @Bean
    public JwtDecoder jwtDecoder(){
        var secretKey = new SecretKeySpec(jwtSecretKey.getBytes(), algorithm: "");
        return NimbusJwtDecoder.withSecretKey(secretKey).macAlgorithm(MacAlgorithm.HS256).build();
    }


    @Bean
    public AuthenticationManager authenticationManager(PersonService personService){
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(personService);
        provider.setPasswordEncoder(new BCryptPasswordEncoder());

        return new ProviderManager(provider);
    }
}
```

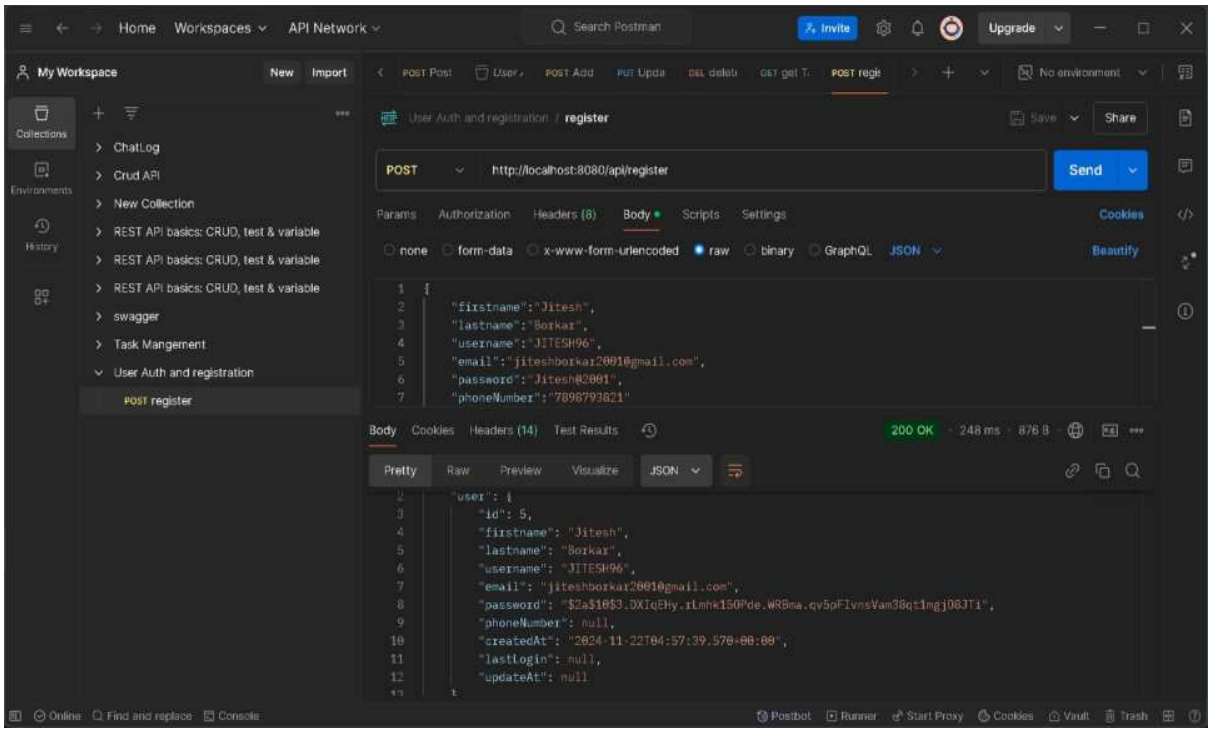
Application properties

```properties
#MySql config
spring.datasource.url = jdbc:mysql://localhost:3306/users
spring.datasource.username = root
spring.datasource.password = Jitesh@2001
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver


#JPA and Hibernate
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto= none
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.security.user.name=Jitesh
spring.security.user.password =Jitesh@2001

#jwt security secretKey
security.jwt.secret-key=e6bb2f3ca9b69683c22e383562566d929d02c4728ca52b72c9e897cf2e280473
security.jwt.issuer=Person
```

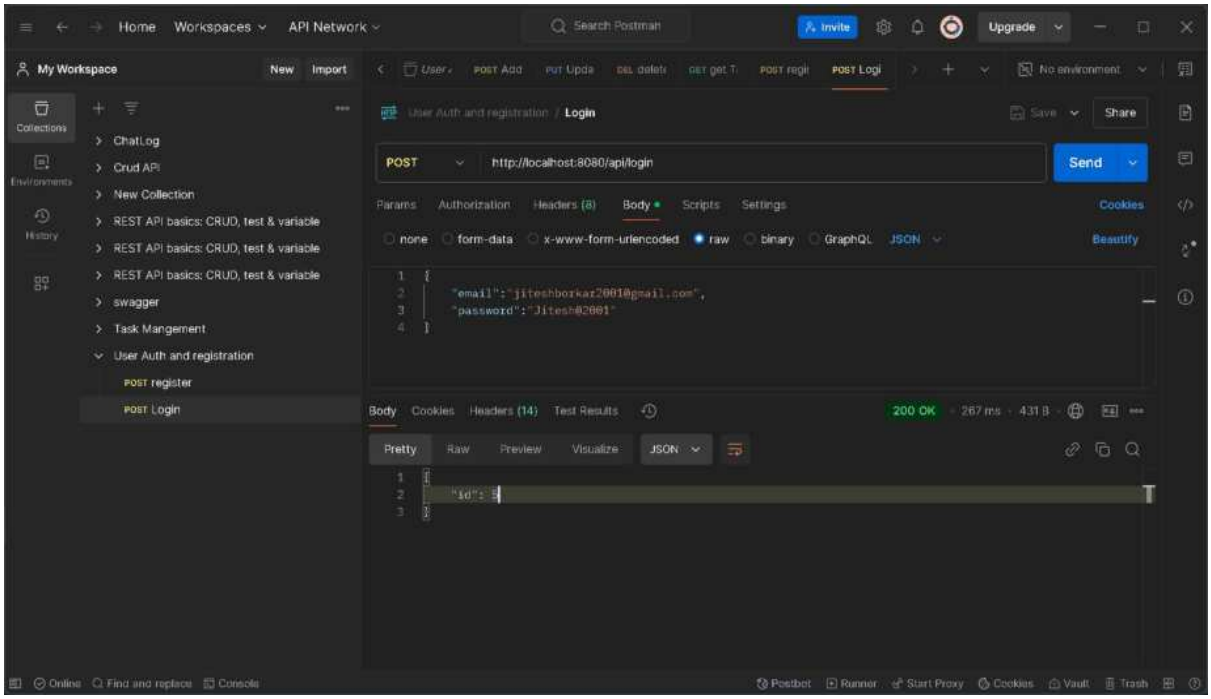
Testing and Validation

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

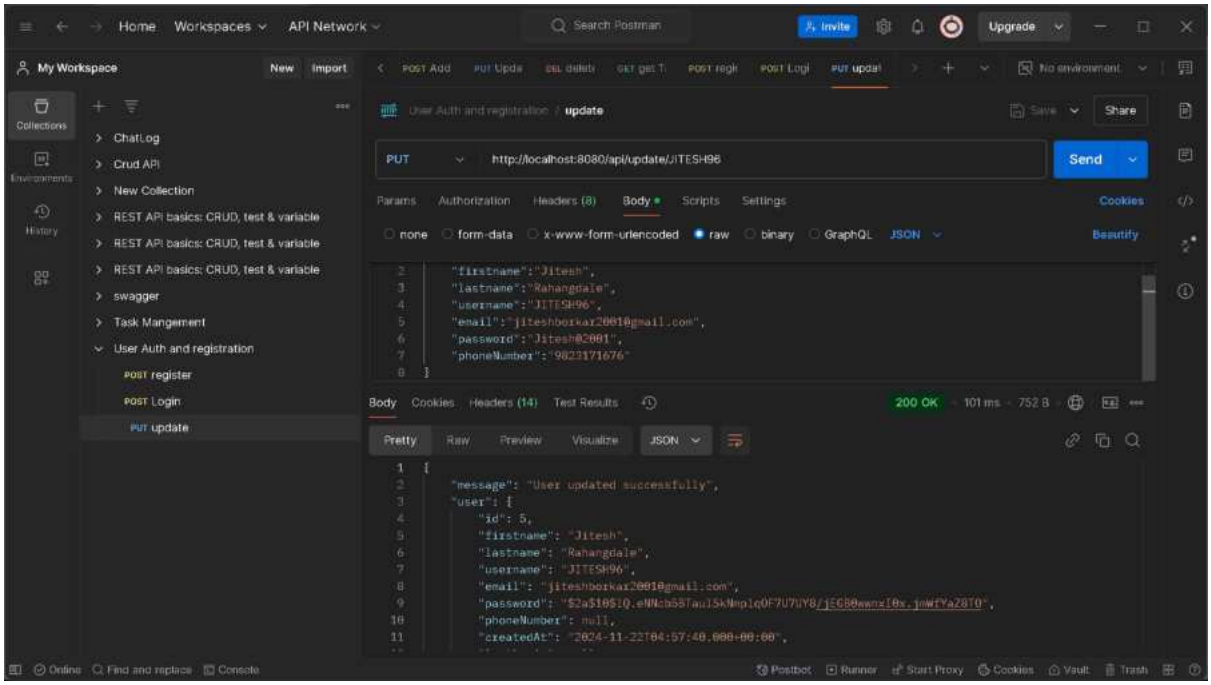
POST api/register



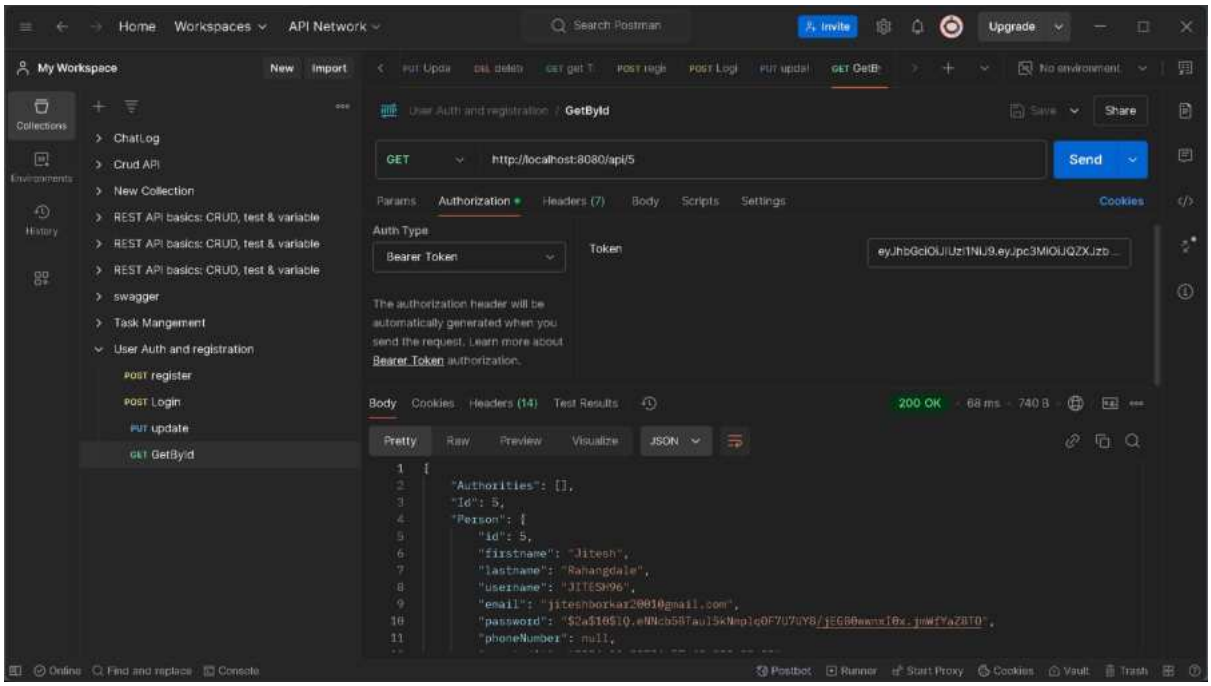
POST api/login



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

→Serive

- 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

```java
package com.Infosys.WellbeingTest.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;
import org.springframework.data.mongodb.core.mapping.Field;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Choice {
    @Field("option")
    private String option;

    @Field("score")
    private int score;
}
```

Class : Question

```java
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Document(collection = "questions")
public class Question {

    @Id
    private String id;
    private String questionText;
    private List<Choice> choices;

}
```

Class : WellbeingTest

```java
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.List;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
@Document(collection = "questions")
public class Question {

    @Id
    private String id;
    private String questionText;
    private List<Choice> choices;

}
```

Repository :

Define JPA repositories for each entity

Class: QuestionRepository

```java
package com.Infosys.WellbeingTest.repository;

import com.Infosys.WellbeingTest.model.Question;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface QuestionRepository extends MongoRepository<Question, String> {
}
```

Class: WellbeingTestRepo

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

```java
import com.Infosys.WellbeingTest.dto.WellbeingTestDTO;
import com.Infosys.WellbeingTest.model.WellbeingTest;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public interface WellbeingService {

    // Method to take a test and return the WellbeingTestRepo object
    WellbeingTest takeTest(WellbeingTestDTO wellbeingTestDTO);

    List<WellbeingTest> getTestResults(int userId);

    List<String> getRecommendations(int id);

    // Method to delete a test by its ID
    boolean deleteTestById(int id);
}
```

Class: WellbeignServiceImpl

```java
@Service
public class WellbeingServiceImpl implements WellbeingService {

    @Autowired
    private QuestionService questionService;

    @Autowired
    private WellbeingTestRepo wellbeingTestRepository;

    @Override
    public WellbeingTest takeTest(WellbeingTestDTO wellbeingTestDTO) {
        AtomicInteger score = new AtomicInteger();
        List<Question> questions = questionService.getAllQuestions();
        Map<String, String> responses = wellbeingTestDTO.getResponses();

        // Check that the responses map is not empty
        if (responses == null || responses.isEmpty()) {
            throw new IllegalArgumentException("Responses cannot be empty");
        }

        // Calculate the score based on responses
        for (Question question : questions) {
```

```java
public class WellbeingServiceImpl implements WellbeingService {
    public WellbeingTest takeTest(WellbeingTestDTO wellbeingTestDTO) {
        // Calculate the score based on responses
        for (Question question : questions) {
            String questionText = question.getQuestionText();
            String answer = responses.get(questionText);  // Get user's response for th

            if (answer != null && !answer.isEmpty()) {
                List<Choice> choices = question.getChoices();  // Get available choices

                if (choices != null && !choices.isEmpty()) {
                    Optional<Choice> selectedChoice = choices.stream()
                            .filter(choice -> choice.getOption().equalsIgnoreCase(answe
                            .findFirst();

                    // Add the score for the selected choice
                    selectedChoice.ifPresent(choice -> {
                        score.addAndGet(choice.getScore());
                        System.out.println("Added score for question: " + questionText
                    });
                } else {
                    System.out.println("No choices available for question: " + question
```

```java
public class WellbeingServiceImpl implements WellbeingService {
    public WellbeingTest takeTest(WellbeingTestDTO wellbeingTestDTO) {
        }
        }

        // Log the final score to verify
        System.out.println("Final calculated score: " + score);

        // Create a WellbeingTest object to save to the database
        WellbeingTest wellbeingTest = new WellbeingTest();
        wellbeingTest.setUserId(wellbeingTestDTO.getUserId());  // Set the user_id
        wellbeingTest.setScore(score.get());
        wellbeingTest.setTaken_at(LocalDateTime.now());
        wellbeingTest.setStatus(determineStatus(score.get()));

        // Save the wellbeingTest object to the repository
        return wellbeingTestRepository.save(wellbeingTest);
    }

    @Override
    public List<WellbeingTest> getTestResults(int userId) {
```

```
> src > main > java > com > Infosys > WellbeingTest > service > WellbeingServiceImpl   124:1   WellbeingTest   One Dark   CRLF   UTF-8   4 spaces
```

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);
```

```java
public class WellbeingTestController {
    public List<WellbeingTest> getTestResults(@PathVariable int userId) {
        return wellbeingService.getTestResults(userId);
    }

    @GetMapping("/getRecommendations/{id}")
    public List<String> getRecommendations(@PathVariable int id) {
        return wellbeingService.getRecommendations(id);
    }

    // Endpoint to delete a test by ID (DELETE request)
    @DeleteMapping("/deleteTest/{id}")
    public ResponseEntity<String> deleteTestById(@PathVariable("id") int id) {
        // Call the service to delete the test
        boolean isDeleted = wellbeingService.deleteTestById(id);

        // Return appropriate response based on whether the deletion was successful
        if (isDeleted) {
            return new ResponseEntity<>("Test deleted successfully.", HttpStatus.OK);
        } else {
            return new ResponseEntity<>("Test not found.", HttpStatus.NOT_FOUND);
        }
    }
}
```

DTO :

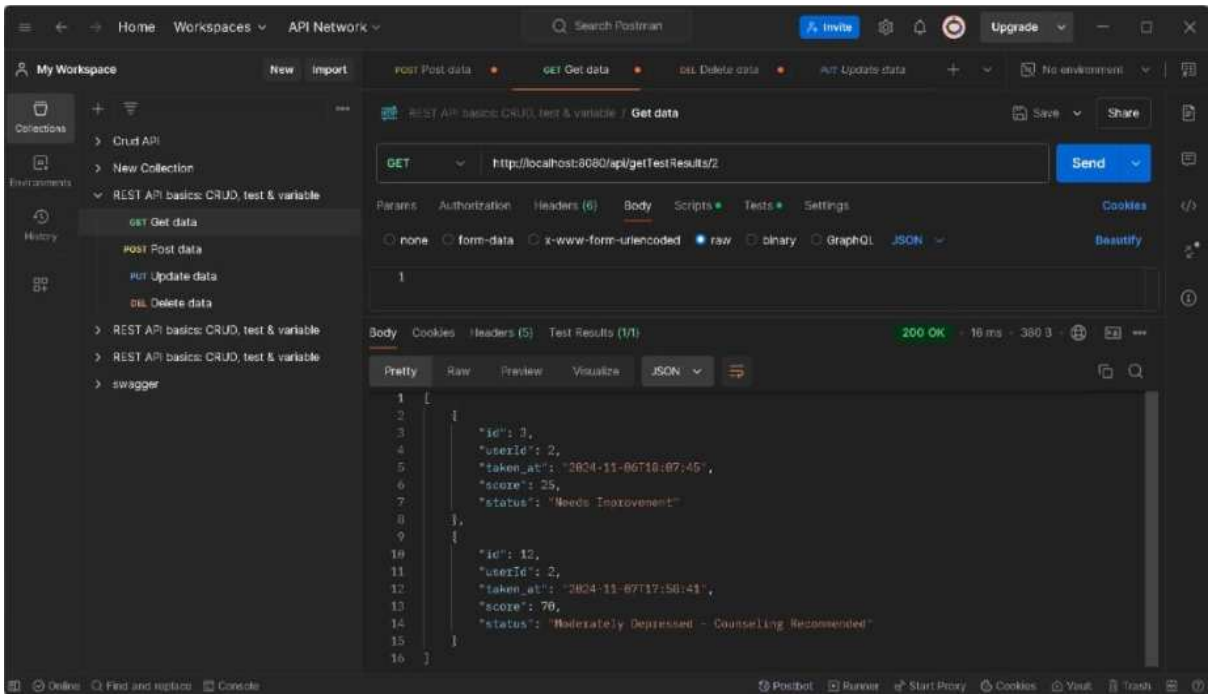
DTOs will help ensure that only the necessary information is sent to the client

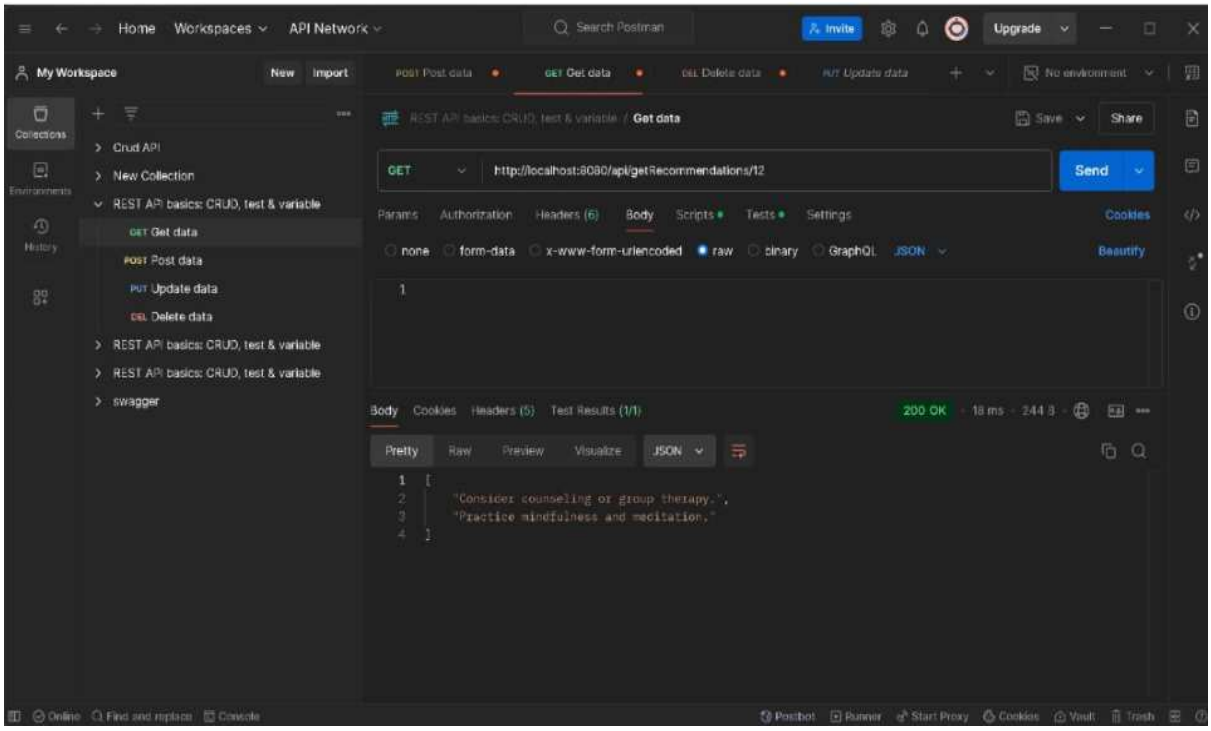
Class : WellbeingTestDTO

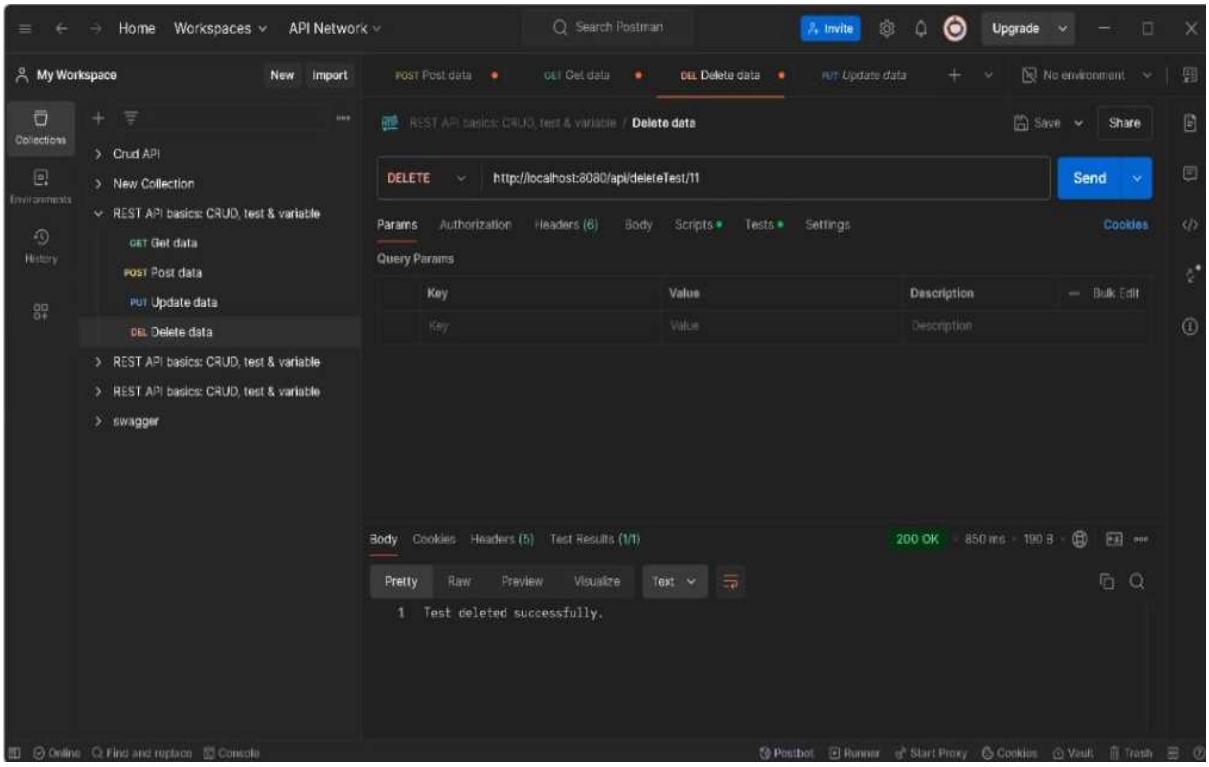
```java
package com.Infosys.WellbeingTest.dto;

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

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


@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class WellbeingTestDTO {
    private int userId;
    private Map<String, String> responses; // Map of question and selected answer

}
```

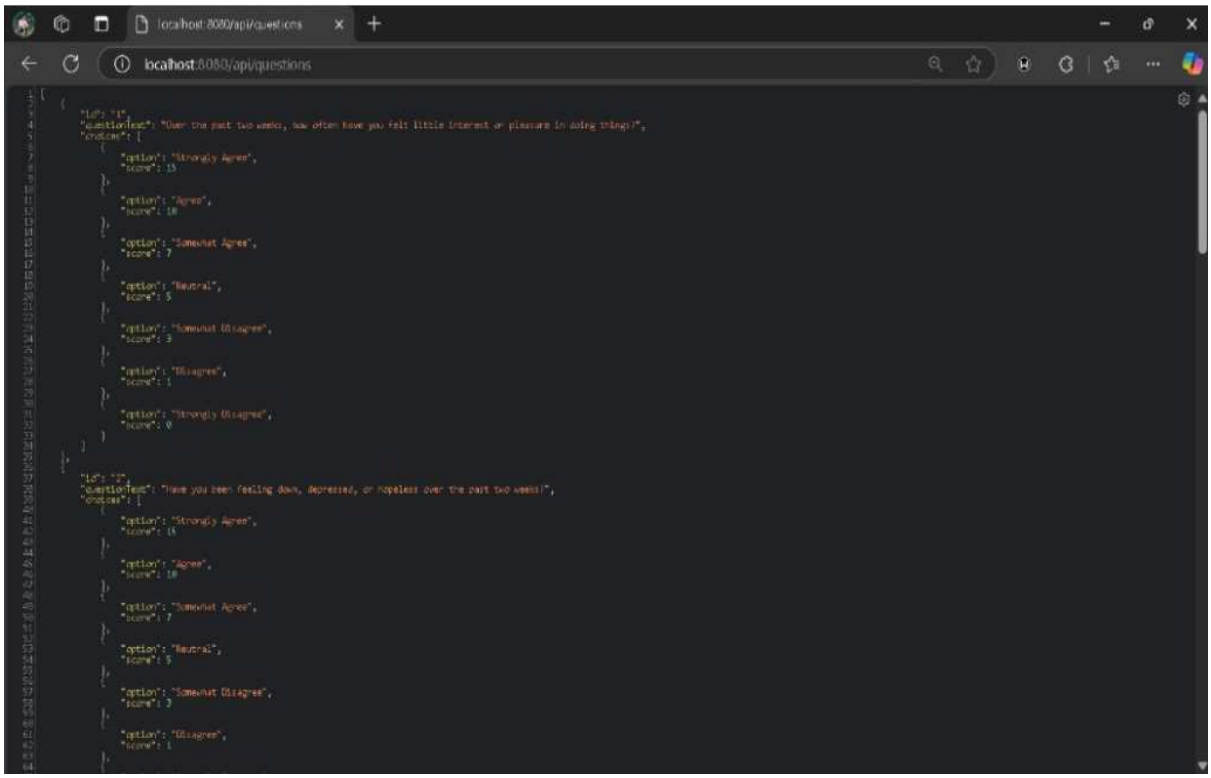
Testing and Validation

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

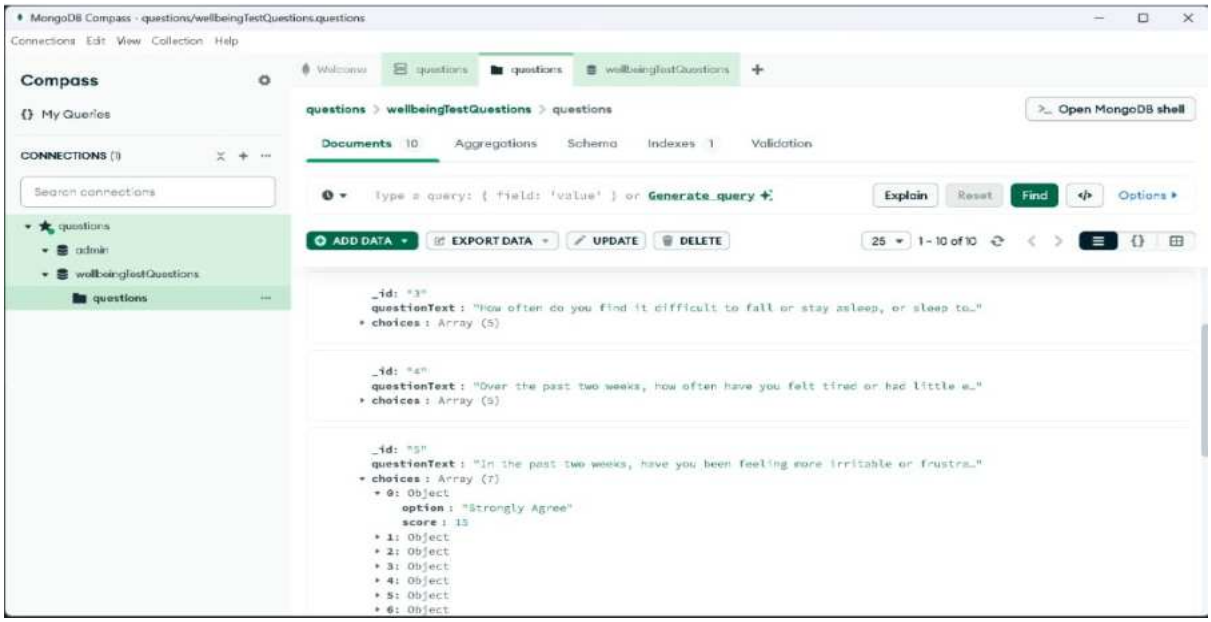




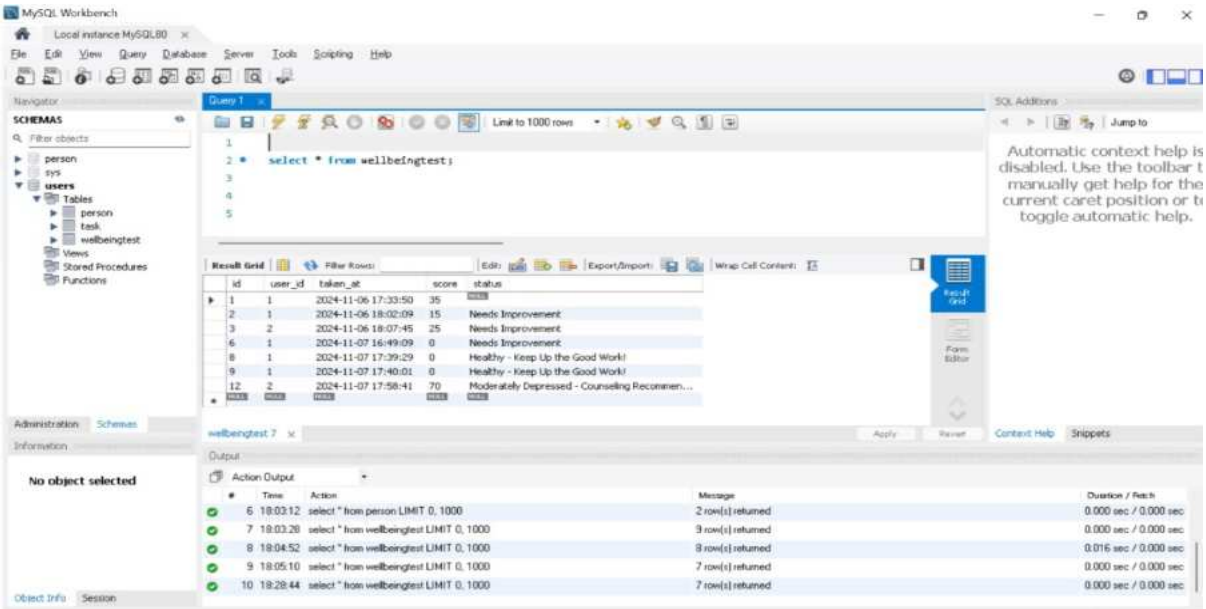

Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
POST Post data
GET Get data
DEL Delete data
PUT Update data
No environment
Collections
Environments
History
Crud API
New Collection
REST API basics: CRUD, test & variable
GET Get data
POST Post data
PUT Update data
DEL Delete data
REST API basics: CRUD, test & variable
REST API basics: CRUD, test & variable
swagger
REST API basics: CRUD, test & variable / Delete data
Save
Share
DELETE
http://localhost:8080/api/deleteTest/11
Send
Params
Authorization
Headers (6)
Body
Scripts
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (5)
Test Results (1/1)
200 OK
850 ms
190 B
Pretty
Raw
Preview
Visualize
Text
1 Test deleted successfully.
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


localhost:8080/api/questions
localhost:8080/api/questions
[
{
"id": "1",
"questionText": "Over the past two weeks, how often have you felt little interest or pleasure in doing things?",
"options": [
{
"option": "Strongly Agree",
"score": 15
},
{
"option": "Agree",
"score": 10
},
{
"option": "Somewhat Agree",
"score": 7
},
{
"option": "Neutral",
"score": 5
},
{
"option": "Somewhat Disagree",
"score": 3
},
{
"option": "Disagree",
"score": 1
},
{
"option": "Strongly Disagree",
"score": 0
}
]
},
{
"id": "2",
"questionText": "Have you been feeling down, depressed, or hopeless over the past two weeks?",
"options": [
{
"option": "Strongly Agree",
"score": 15
},
{
"option": "Agree",
"score": 10
},
{
"option": "Somewhat Agree",
"score": 7
},
{
"option": "Neutral",
"score": 5
},
{
"option": "Somewhat Disagree",
"score": 3
},
{
"option": "Disagree",
"score": 1
},

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

→Serive

- ChatLogService
- ChatLogServiceImpl

Model :

Define models to represent chatbot's messages.

Class: ChatLog

```java
@Data
@Entity
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Table(name = "chatLog")
public class ChatLog {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "user_id", nullable = false)
    private int userId;

    @Column(name = "query")
    private String query;

    @Column(name = "response")
    private String response;

    @Column(name = "timestamp")
    private LocalDateTime timestamp;
}
```

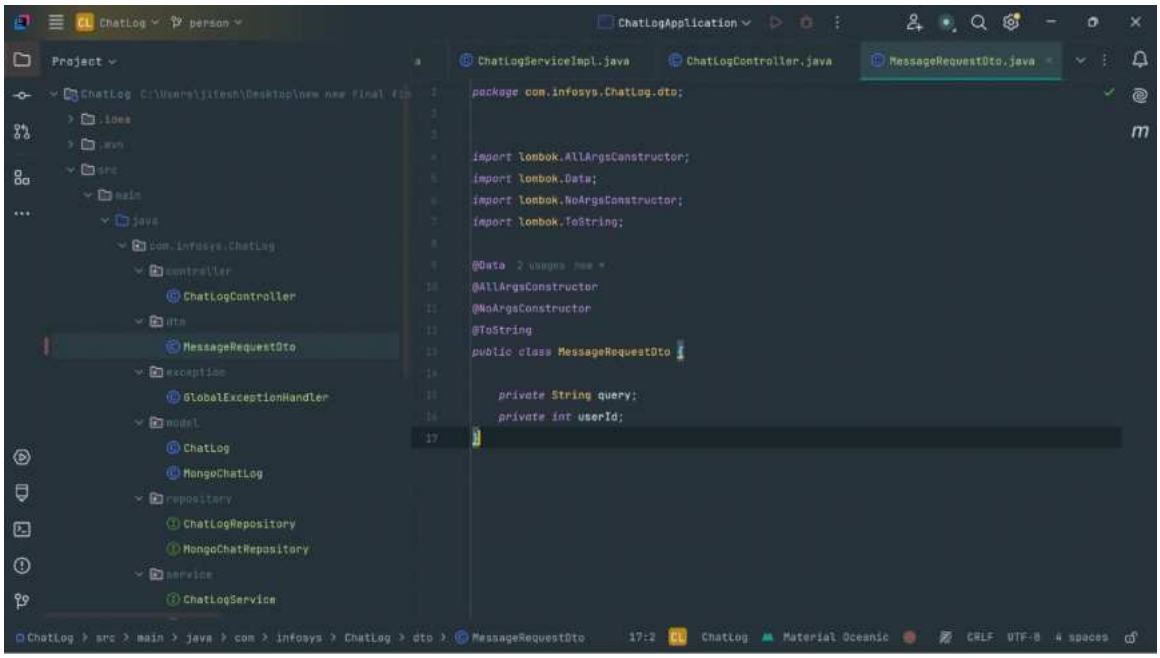
Class : MongoChatLog

```java
package com.infosys.ChatLog.model;

import jakarta.persistence.Entity;
import jakarta.persistence.Id;
import lombok.*;
import org.springframework.data.mongodb.core.mapping.Document;

import java.time.LocalDateTime;

@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
@Document(collection = "chatQueries")
public class MongoChatLog {

    @Id
    private String id;
    private String query;
    private String response;
    private LocalDateTime timestamp;
}
```

DTOs:

Define DTOs to control data sent to the client.

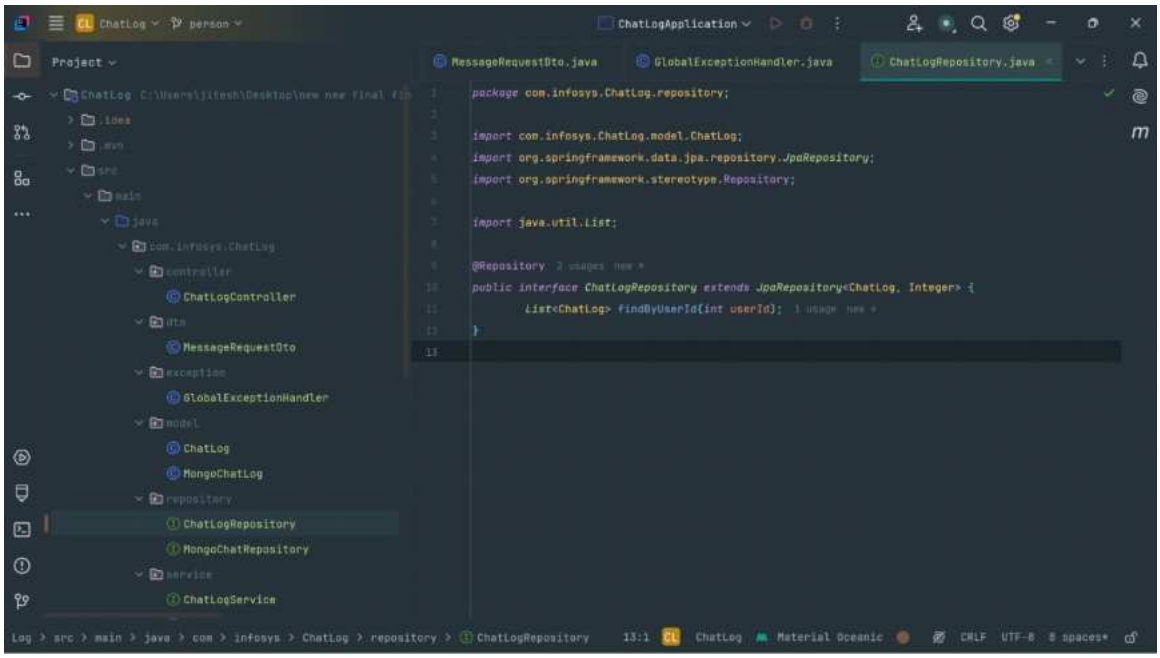
Class: MessageRequestDTO



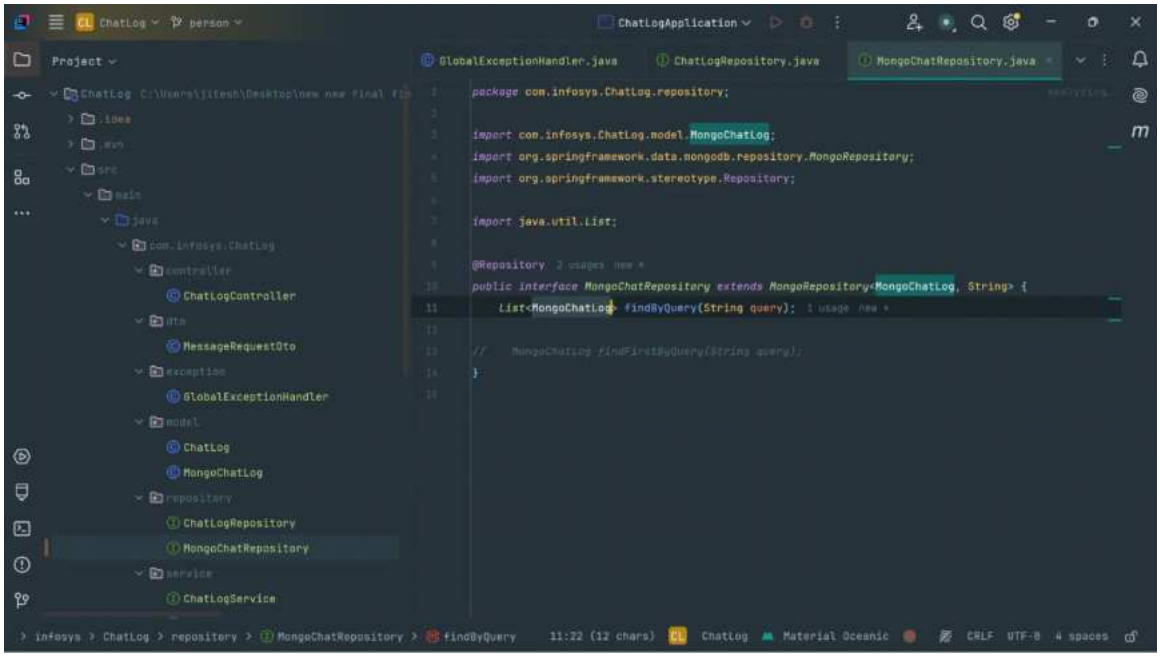
Repository :

Create Repositories to manage entities

Class: ChatLogRepository



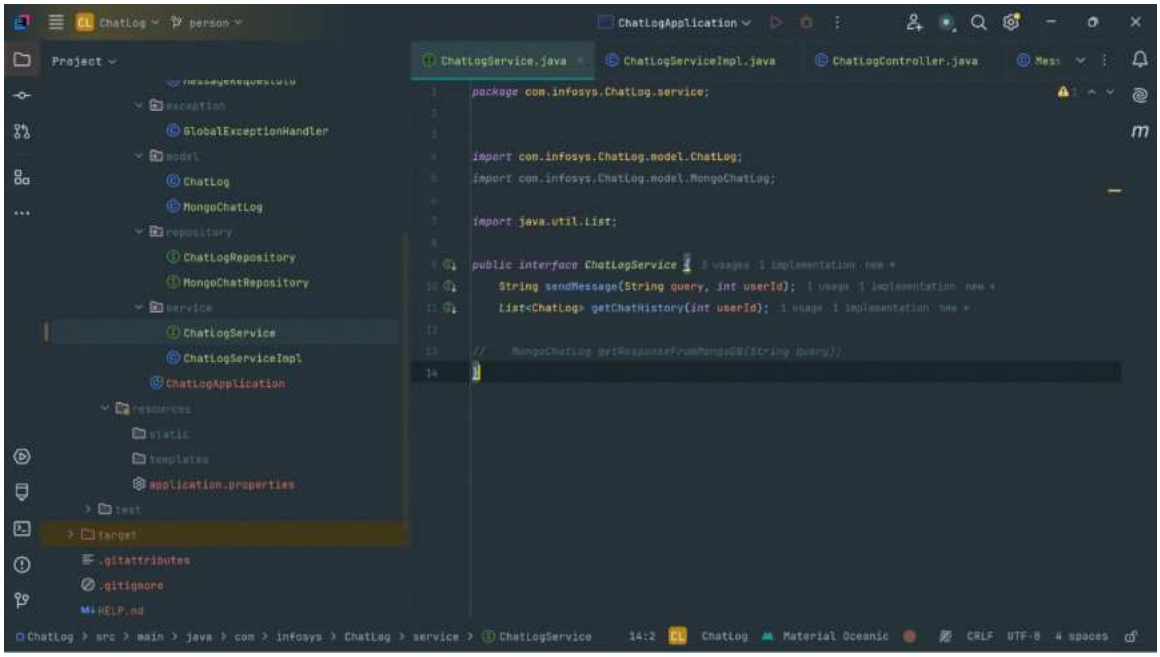
Class: MongoChatLog



Service:

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

Class: ChatLogService



Class: ChatLogServiceImpl

```java
@Service
public class ChatLogServiceImpl implements ChatLogService {

    private static final Logger logger = Logger.getLogger(ChatLogServiceImpl.class.getName());

    @Autowired
    private ChatLogRepository chatLogRepository;

    @Autowired
    private MongoChatRepository mongoChatRepository;

    @Override
    public String sendMessage(String query, int userId) {
        String response = null;
        List<MongoChatLog> mongoChatLogs = null; // Declared outside the try block

        try {
            // Retrieve the response from MongoDB
            mongoChatLogs = mongoChatRepository.findByQuery(query);

            if (mongoChatLogs != null && !mongoChatLogs.isEmpty()) {
                MongoChatLog mongoChatLog = mongoChatLogs.get(0);  // Handle multiple results if a
                response = mongoChatLog.getResponse();
                logger.info( msg: "Response found in MongoDB for query: " + query);
```

```java
    public class ChatLogServiceImpl implements ChatLogService {
        public String sendMessage(String query, int userId) {
            if (mongoChatLogs != null && !mongoChatLogs.isEmpty()) {
                MongoChatLog mongoChatLog = mongoChatLogs.get(0);  // Handle multiple results if a
                response = mongoChatLog.getResponse();
                logger.info( msg: "Response found in MongoDB for query: " + query);
            } else {
                response = "This is a mocked AI response for: " + query;  // Mock response if not
                logger.info( msg: "No response found in MongoDB, using mock response.");
            }

            // Save the chat log to MySQL
            ChatLog chatLog = new ChatLog();
            chatLog.setUserId(userId);
            chatLog.setQuery(query);
            chatLog.setResponse(response);
            chatLog.setTimestamp(LocalDateTime.now());

            chatLogRepository.save(chatLog);
            logger.info( msg: "ChatLog saved successfully to MySQL.");
        } catch (Exception e) {
            logger.severe( msg: "Error while processing sendMessage for query: " + query + ". Error
        }
```

```java
    public class ChatLogServiceImpl implements ChatLogService {
        public String sendMessage(String query, int userId) {
                mongoChatRepository.save(newMongoChatLog);
                logger.info( msg: "Query and response successfully saved to MongoDB for query: " +
            }
        } catch (Exception e) {
            logger.severe( msg: "Error saving query and response to MongoDB for query: " + query +
        }

        return response;
    }

    @Override
    public List<ChatLog> getChatHistory(int userId) {
        try {
            return chatLogRepository.findByUserId(userId);
        } catch (Exception e) {
            logger.severe( msg: "Error retrieving chat history for userId " + userId + ": " + e.ge
            return null;
        }
    }
}
```

Controller :

Create REST endpoints for Chatbot interaction.

Class: ChatLogController

```java
@RestController
@RequestMapping("/api")
public class ChatLogController {

    private static final Logger logger = Logger.getLogger(ChatLogController.class.getName());

    @Autowired
    private ChatLogService chatLogService;

    // Endpoint to send a message and get a response
    @PostMapping("/send")
    public String sendMessage(@RequestBody MessageRequestDto messageRequestDto) {
        try {
            return chatLogService.sendMessage(messageRequestDto.getQuery(), messageRequestDto.get
        } catch (Exception e) {
            logger.severe( msg: "Error in /send endpoint: " + e.getMessage());
            return "An error occurred while processing the message.";
        }
    }

    // Endpoint to get chat history for a user
    @GetMapping("/history/{userId}")
    public List<ChatLog> getChatHistory(@PathVariable int userId) {
        try {
```

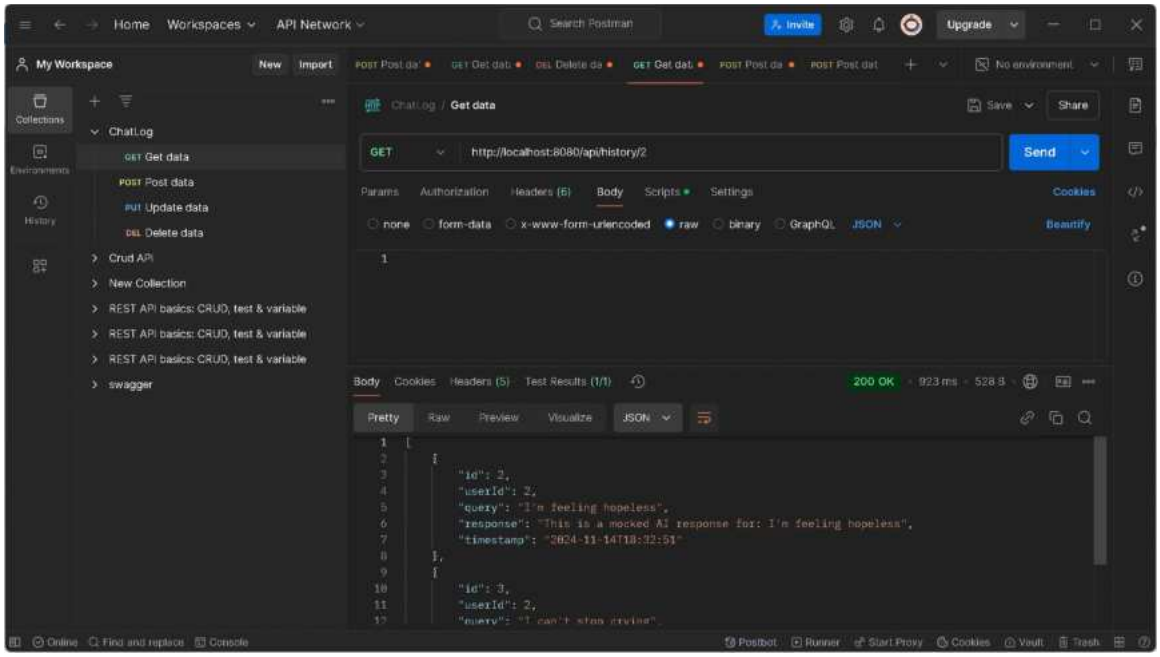
```java
public class ChatLogController {
    public String sendMessage(@RequestBody MessageRequestDto messageRequestDto) {
            logger.severe( msg: "Error in /send endpoint: " + e.getMessage());
            return "An error occurred while processing the message.";
        }
    }

    // Endpoint to get chat history for a user
    @GetMapping("/history/{userId}")
    public List<ChatLog> getChatHistory(@PathVariable int userId) {
        try {
            return chatLogService.getChatHistory(userId);
        } catch (Exception e) {
            logger.severe( msg: "Error in /history endpoint: " + e.getMessage());
            return null;
        }
    }
}
```

Testing and Validation

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

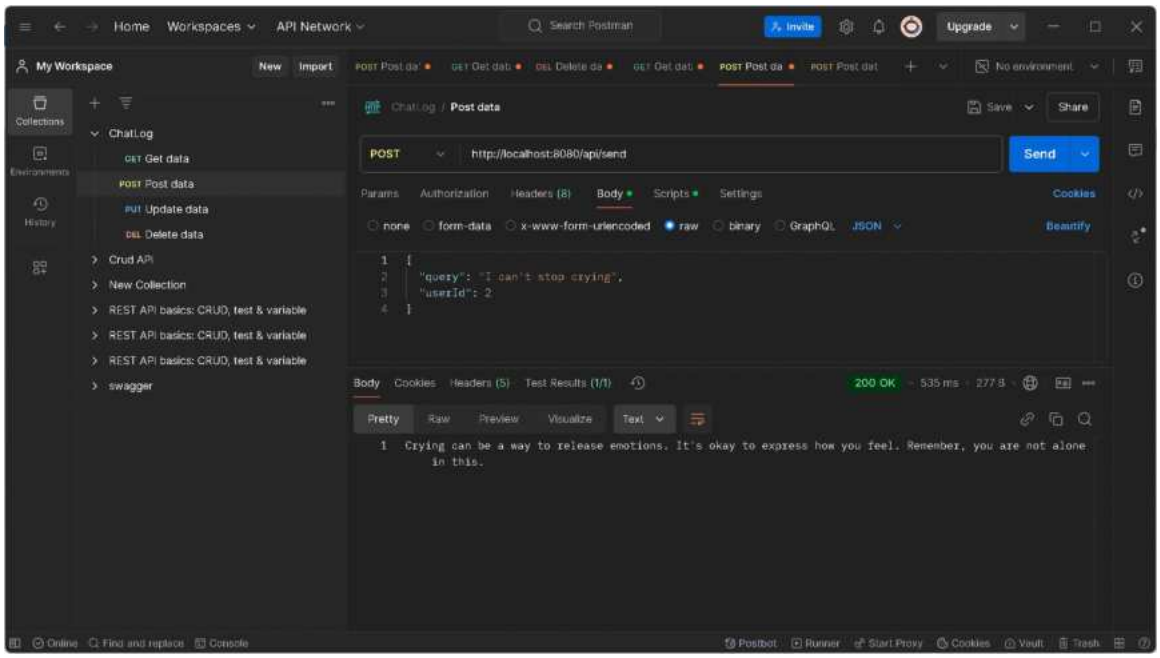
GET

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

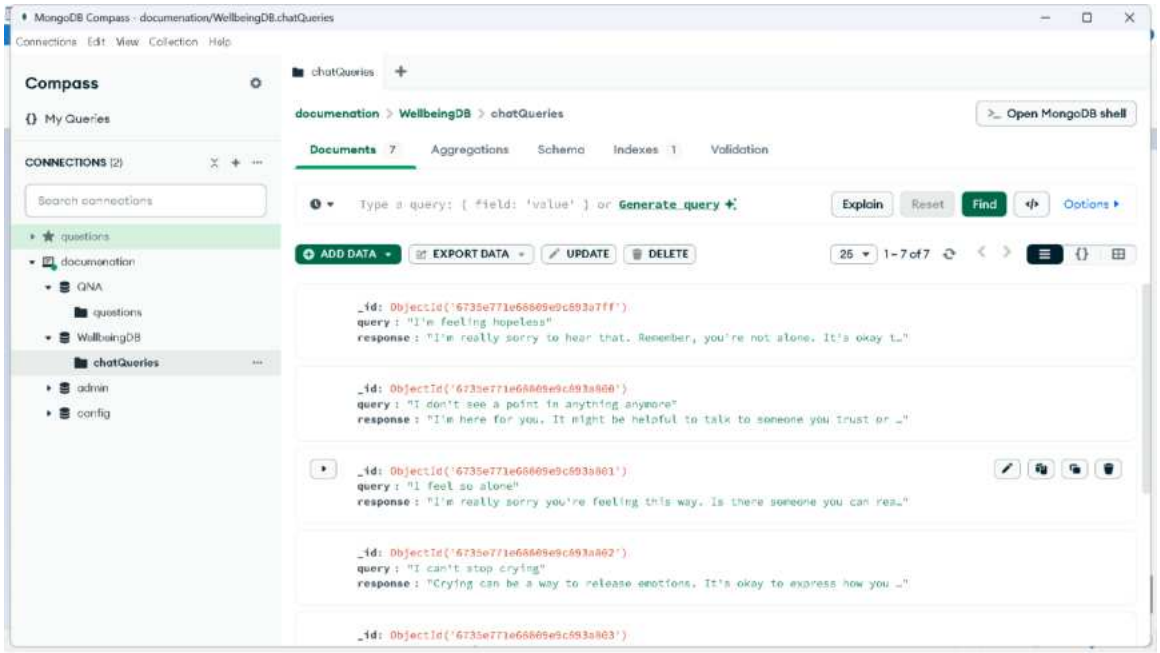


POST

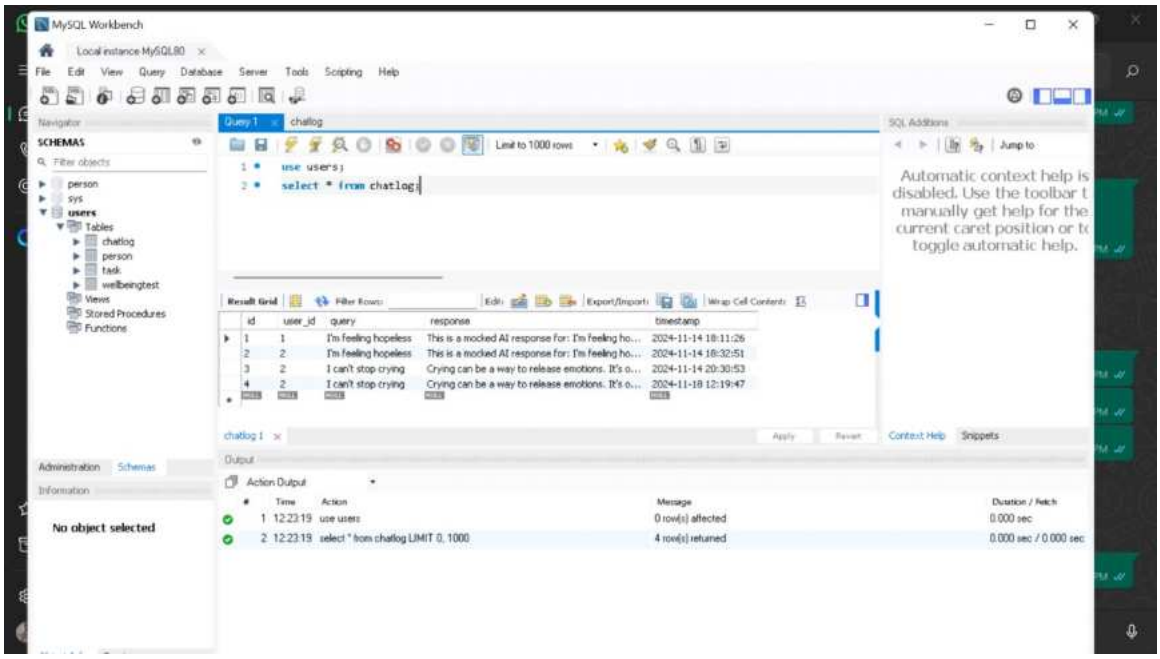
http://localhost:8080/api/send



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

→Serive

- TaskService
- TaskServiceImpl

Model:

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

Class: Task

```java
@NoArgsConstructor
@ToString
@Table(name = "Task")
public class Task {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "user_id" , nullable= false)
    private int userId;

    @Column(name="priority" , nullable = false)
    private String priority;

    private String title;
    @Column(name = "created_at")
    private LocalDateTime createdAt;
    @Column(name = "due_date")
    private LocalDateTime dueDate;

    @Column(name = "is_completed")
    private boolean isCompleted;
```

Controller:

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

Class: TaskController

```java
import java.util.List;

@RestController
@RequestMapping("/api")
@CrossOrigin("http://localhost:3000")
public class TaskController {

    @Autowired
    private TaskService taskService;

    // Create Task
    @PostMapping("/addTask")
    public ResponseEntity<?> createTask(@RequestBody TaskDto taskDto) {
        try {
            Task createdTask = taskService.createTask(taskDto);
            return ResponseEntity.status(HttpStatus.CREATED).body(createdTask);
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Task creation failed");
        }
    }

    // Update Task
```

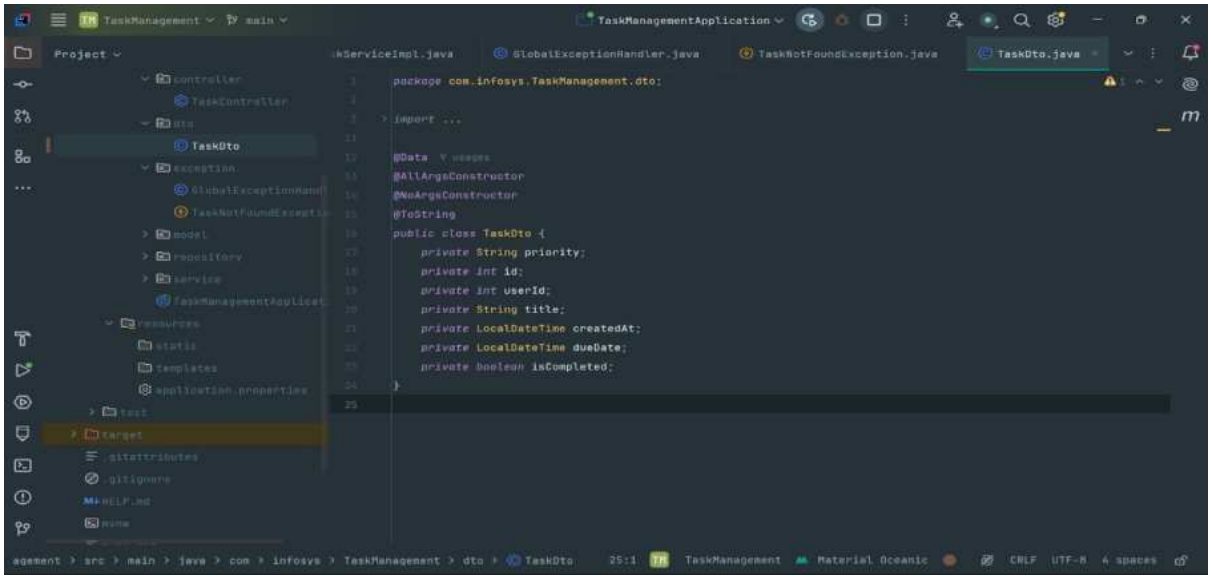
```java
public class TaskController {
    // Update Task
    @PutMapping("/updateTask")  no usages
    public ResponseEntity<?> updateTask(@RequestBody TaskDto taskDto) {
        try {
            Task updatedTask = taskService.updateTask(taskDto);
            return ResponseEntity.ok(updatedTask);
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Task update failed");
        }
    }

    // Delete Task
    @DeleteMapping("/deleteTask/{id}")  no usages
    public ResponseEntity<?> deleteTask(@PathVariable("id") int id) {
        try {
            taskService.deleteTask(id);
            return ResponseEntity.ok().build();
        } catch (IllegalArgumentException e) {
            return ResponseEntity.badRequest().body(e.getMessage());
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Task deletion failed");
```

```java
public class TaskController {
    // Get Tasks by User ID
    @GetMapping("/{userId}")  no usages
    public ResponseEntity<?> getTasksByUserId(@PathVariable int userId) {
        try {
            List<Task> tasks = taskService.getTask(userId);
            if (tasks.isEmpty()) {
                return ResponseEntity.status(HttpStatus.NOT_FOUND).body("No tasks found for the given user ID");
            }
            return ResponseEntity.ok(tasks);
        } catch (Exception e) {
            return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Failed to fetch tasks");
        }
    }

    // Mark Task as Completed
    @PatchMapping("/{id}/complete")  no usages
    public ResponseEntity<?> markTaskCompleted(@PathVariable int id) {
        try {
            Task completedTask = taskService.markTaskCompleted(id);
            return ResponseEntity.ok(completedTask);
        } catch (IllegalArgumentException e) {
            return ResponseEntity.status(HttpStatus.NOT_FOUND).body(e.getMessage());
        } catch (Exception e) {
```

Dto:

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

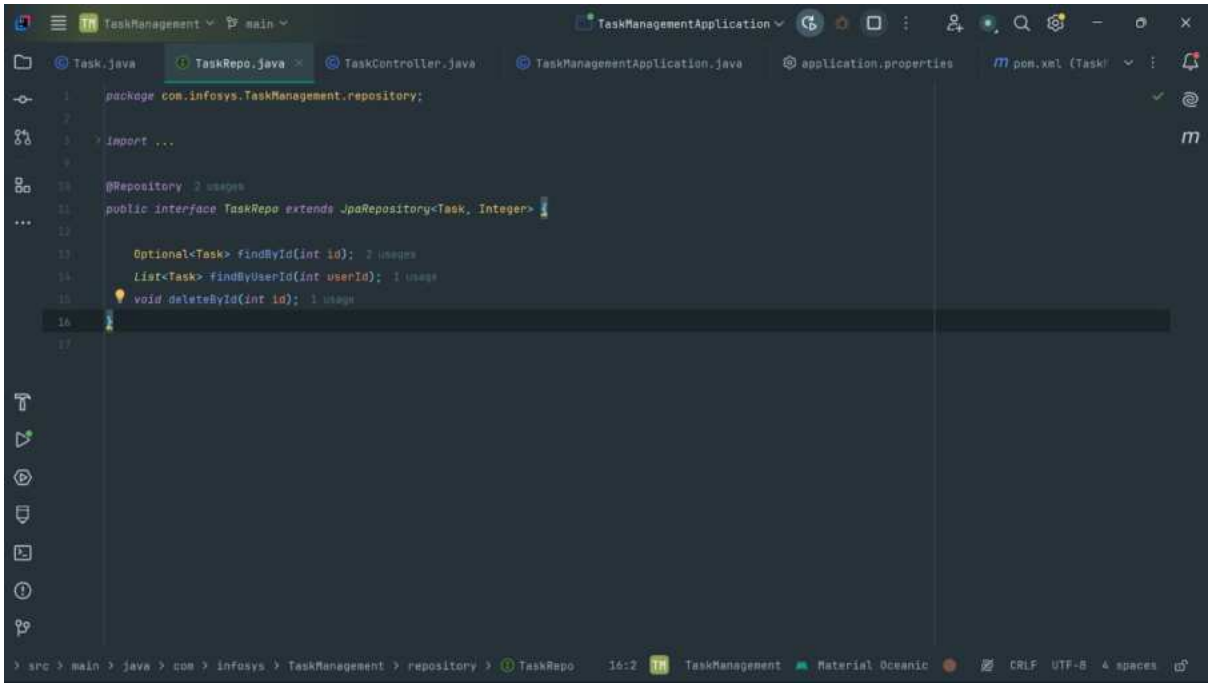
Class: TaskDto



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

```java
package com.infosys.TaskManagement.service;

import ...


@Service
public interface TaskService {

    Task createTask(TaskDto taskDto);

    Task updateTask(TaskDto taskDto);

    void deleteTask(int id);

    List<Task> getTask(int userId);

    Task markTaskCompleted(int id);


}
```

Class: TaskServiceImpl

```java
package com.infosys.TaskManagement.service;

import ...

@Service
public class TaskServiceImpl implements TaskService{

    @Autowired
    private TaskRepo taskRepo;

    @Override
    public Task createTask(TaskDto taskDto) {
        System.out.print(taskDto.getUserId());
        Task task = new Task();
        task.setUserId(taskDto.getUserId());
        task.setTitle(taskDto.getTitle());
        task.setPriority(taskDto.getPriority());
        task.setCreatedAt(LocalDateTime.now());
        task.setDueDate((taskDto.getDueDate()));
        task.setCompleted(false);
        return taskRepo.save(task);
    }

    @Override
    public Task updateTask(TaskDto taskDto) {
```

```java
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(task.getPriority());
        task.setDueDate(taskDto.getDueDate());
        task.setPriority(taskDto.getPriority());
        task.setCompleted(taskDto.isCompleted());
        return taskRepo.save(task);
    }

    @Override 1 usage
    public void deleteTask(int id) {
        System.out.print("id is : "+id+"\n");
        if (!taskRepo.existsById(id)) {
            throw new TaskNotFoundException("Task not found with id: " + id);
        }
        taskRepo.deleteById(id);
    }

    @Override 1 usage
    public List<Task> getTask(int userId) { return taskRepo.findByUserId(userId); }
```

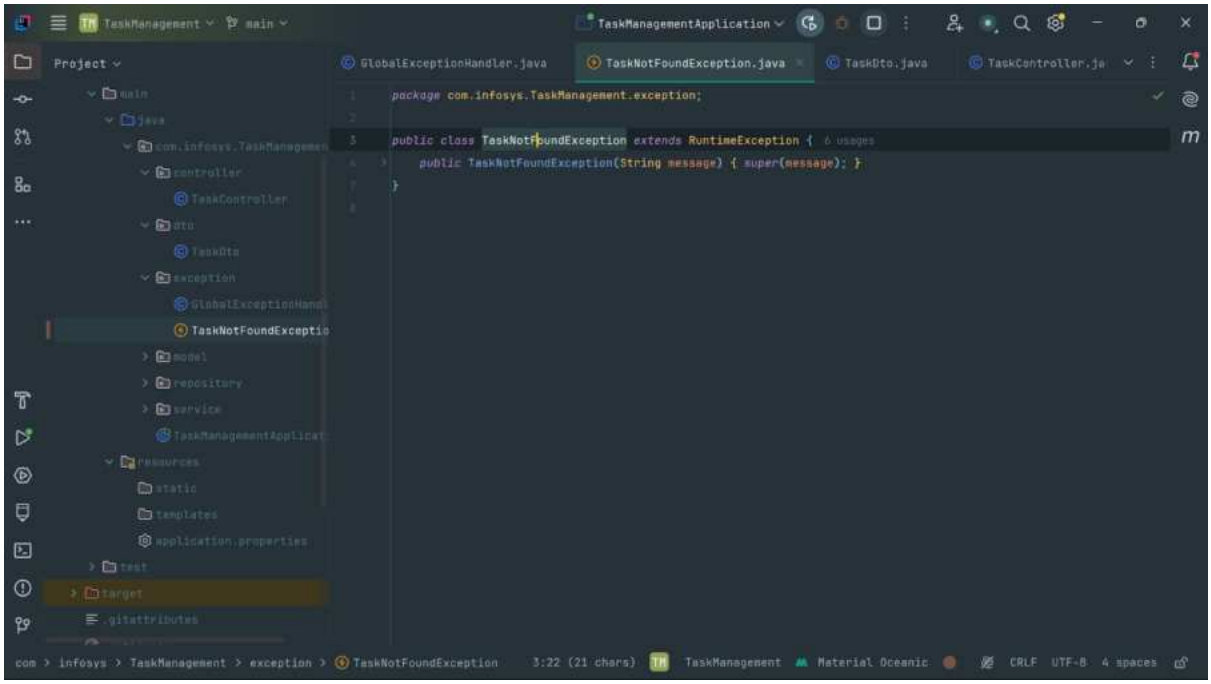
Exception:

Defines custom error scenarios.

Class: GlobalExceptionHandler

```java
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.getMessag
    }
}
```

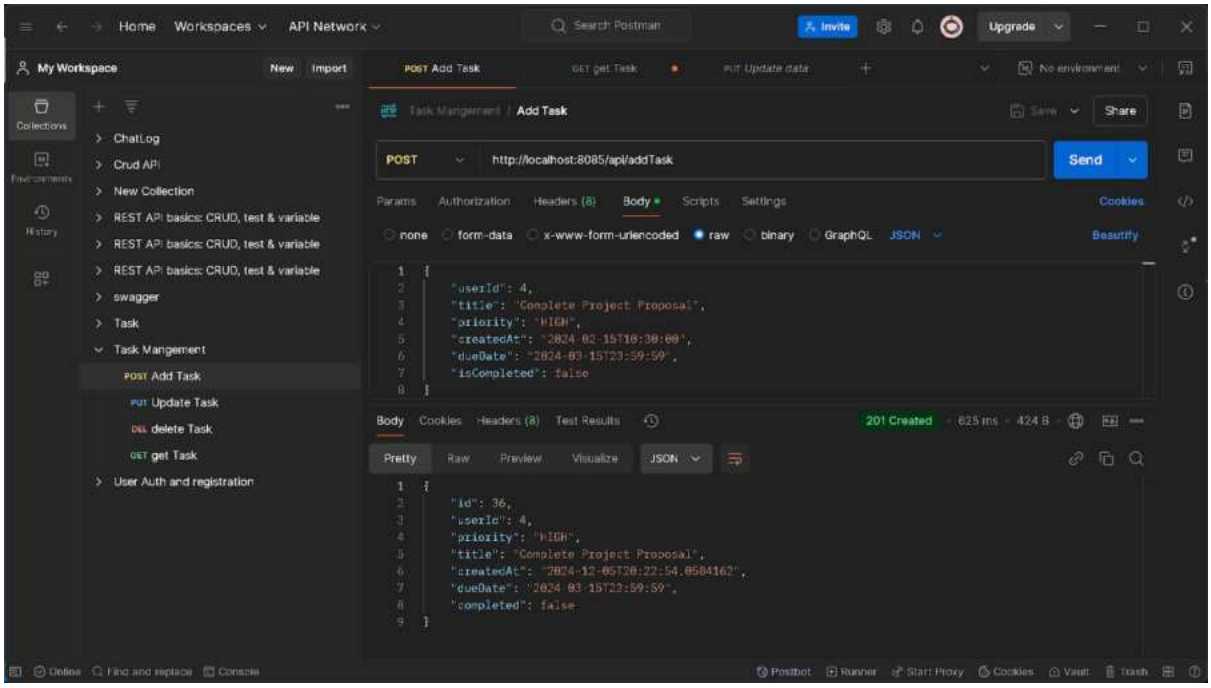
Class: TaskNotFoundException



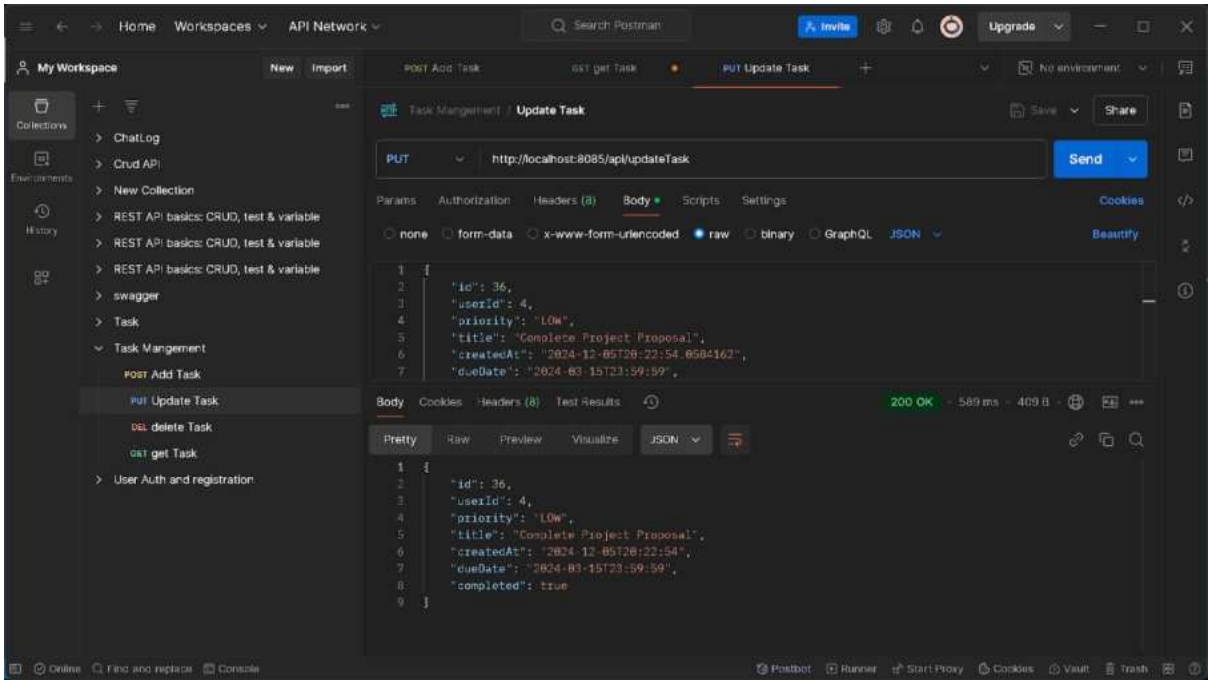
Testing and Validation

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

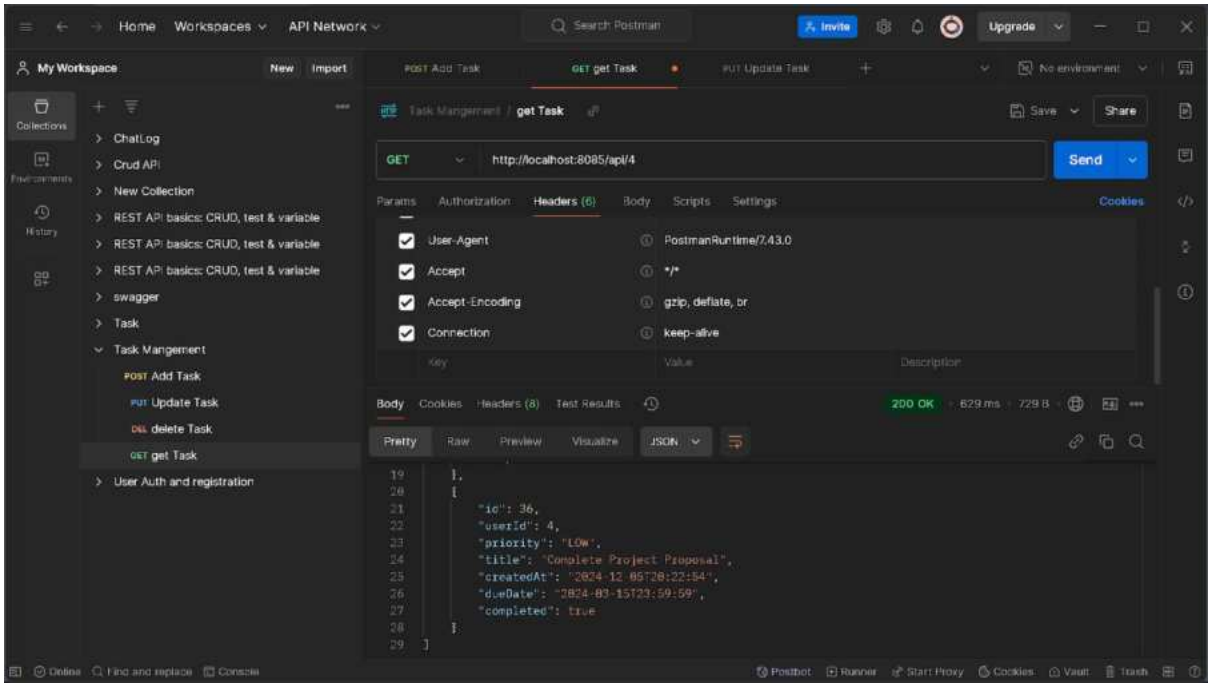
POST: api/addTask



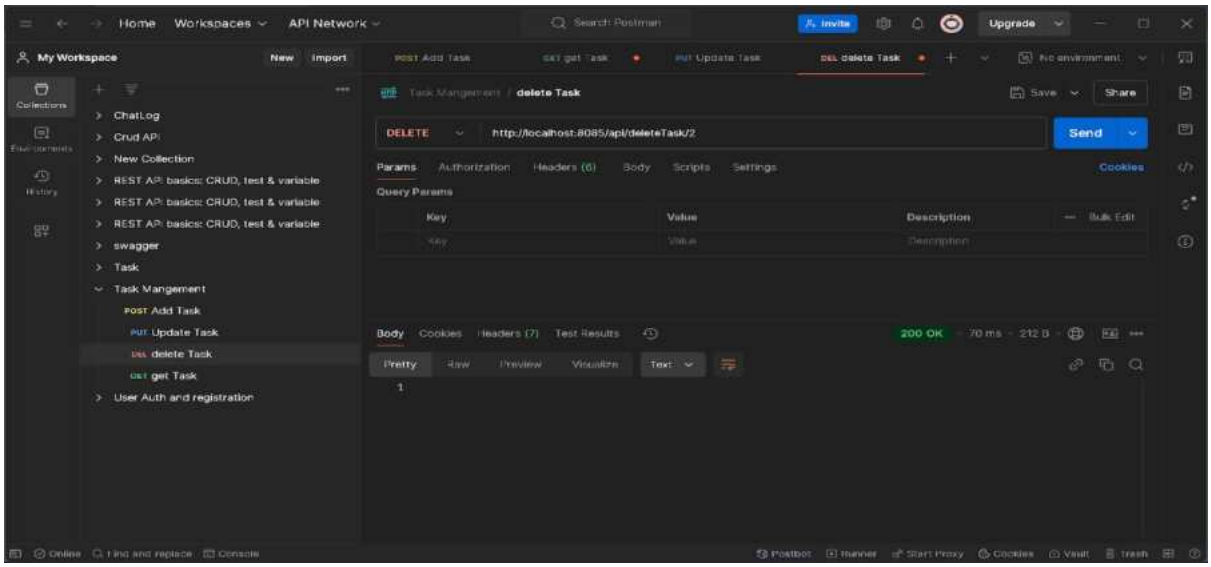
PUT: api/updateTask



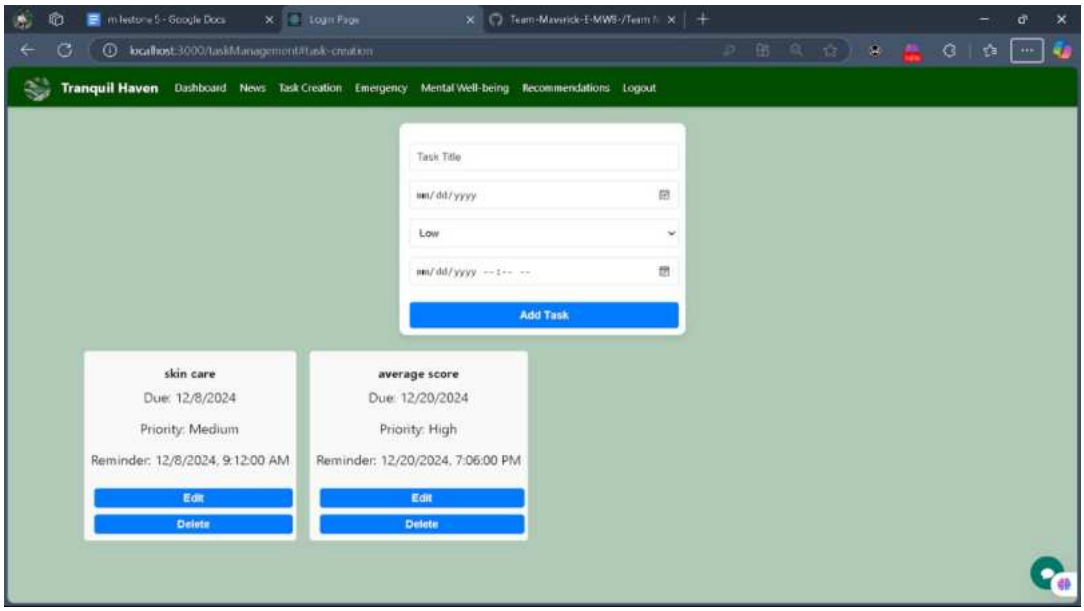
GET : api/{userId}



DELETE : api/deleteTask/{id}



Output



Conclusion:

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