Tender Management API

In this hands-on, you need to create a REST Api in Spring boot, which is used to manage the bidding details and get approved.

Models:

RoleModel:

Field Name	Datatype	Primary Key	Foreign Key	Comments
id	Integer	Yes		autoincrement
rolename	String			Unique

UserModel:

Field Name	Datatype	Primary Key	Foreign Key	Comments
id	Integer	Yes		autoincrement
username	String			
companyName	String			
email	String			Unique
password	String			
role	Integer		Yes	

BiddingModel:

Field Name	Datatype	Primary Key	Foreign Key	Comments
id	Integer	Yes		autoincrement
biddingId	Integer			Unique
projectName	String			projectName is a final string with value "Metro Phase V 2024"
bidAmount	Double			
yearsToComplete	Double			
dateOfBidding	String			Current date in dd/MM/yyyy format
status	String			Default value = "pending"
bidderId	Integer		Yes	

If any of the above validations are failing, return with a response code of 400 - Bad Request

Endpoints:

1.POST METHOD - **/login**

Authenticates and creates JWT token with respective authorization

Request Parameters	**Success Response**	**Error Response**

JSON Body -
{
"email":"bidderemail@gmail.com",
"password":"bidder123$"
}
	200 OK
	{
	"jwt":"your_jwt_token",
	"status":200
	}
		400 Bad Request on invalid credentials

2.POST METHOD - **/bidding/add**

Adds a new Bidding. Note: **bidderId** should point to the bidder who created the bidding.

Request Parameters	**Success Response**	**Error Response**

JSON Body -
{
"biddingId":2608,
"bidAmount":14000000.0,
"yearsToComplete":2.6
}
	201 CREATED
	{
	"id": 1,
	"biddingId": 2608,
	"projectName": "Metro Phase V 2024",
	"bidAmount": 1.4E7,
	"yearsToComplete": 2.6,
	"dateOfBidding": "07/07/2023",
	"status": "pending",
	"bidderId": 1
	}
		400 Bad Request on invalid credentials

We have initialized the database with the following data

Role

id	**rolename**
1	BIDDER
2	APPROVER

User

username	**companyName**	**password**	**email**	**role**
bidder1	companyOne	bidder123$	bidderemail@gmail.com	1
bidder2	companyTwo	bidder789$	bidderemail2@gmail.com	1
approver	defaultCompany	approver123$	approveremail@gmail.com	2

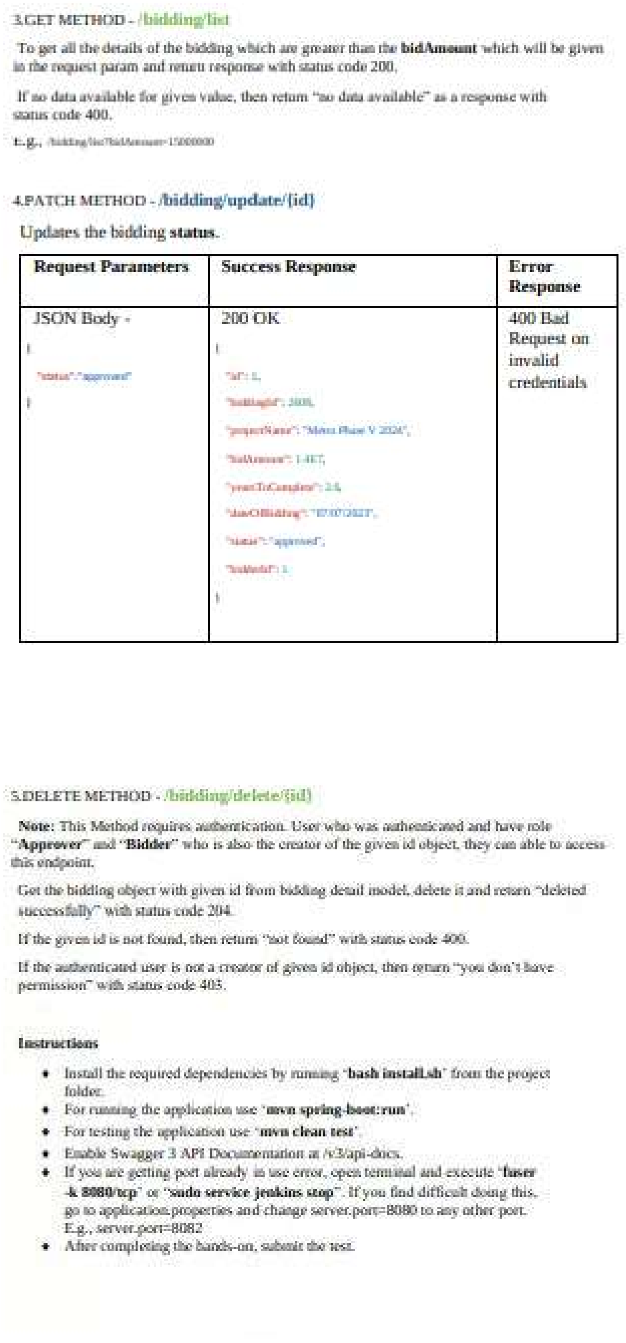
Implement JWT based authorization and authentication with the two above mentioned roles. BIDDER and APPROVER should be identified from the JWT token

JWT token should be sent as a Bearer token in Authorization request header. For example: Authorization value would be Bearer <SPACE><JWT TOKEN>

End-Points Marked in

- Red is accessible only by bidders
- Blue is accessible only by approver
- Green is accessible by both bidder and approver

All other endpoints except **/login** should be authenticated and authorized with the above conditions.



3.GET METHOD - /bidding/list

To get all the details of the bidding which are greater than the **bidAmount** which will be given in the request param and return response with status code 200.

If no data available for given value, then return "no data available" as a response with status code 400.

E.g., /bidding/list?bidAmount=15000000

4.PATCH METHOD - /bidding/update/{id}

Updates the bidding **status**.

Request Parameters	**Success Response**	**Error Response**

JSON Body -
{
"status":"approved"
}
	200 OK
	{
	"id": 1,
	"biddingId": 2608,
	"projectName": "Metro Phase V 2024",
	"bidAmount": 1.4E7,
	"yearsToComplete": 2.6,
	"dateOfBidding": "07/07/2023",
	"status": "approved",
	"bidderId": 1
	}
		400 Bad Request on invalid credentials

5.DELETE METHOD - /bidding/delete/{id}

Note: This Method requires authentication. User who was authenticated and have role "**Approver**" and "**Bidder**" who is also the creator of the given id object, they can able to access this endpoint.

Get the bidding object with given id from bidding detail model, delete it and return "deleted successfully" with status code 204.

If the given id is not found, then return "not found" with status code 400.

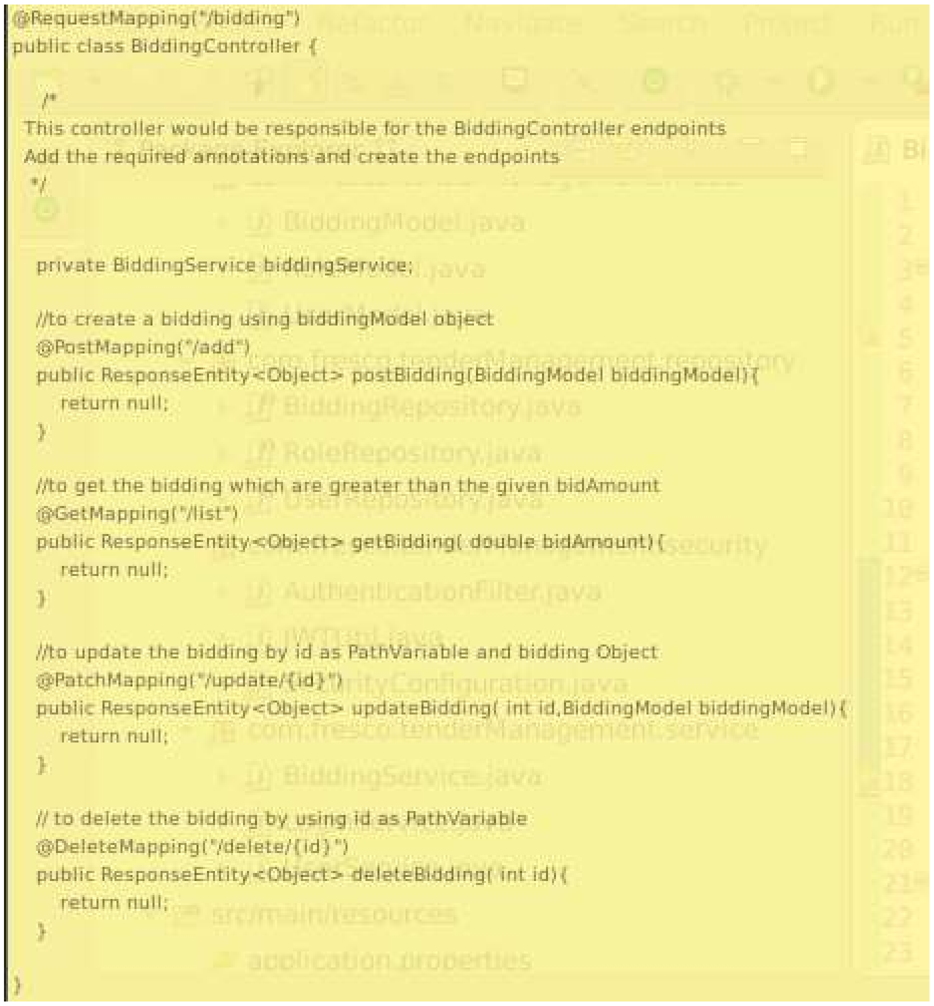
If the authenticated user is not a creator of given id object, then return "you don't have permission" with status code 403.

Instructions

- Install the required dependencies by running '**bash install.sh**' from the project folder.
- For running the application use '**mvn spring-boot:run**'.
- For testing the application use '**mvn clean test**'.
- Enable Swagger 3 API Documentation at /v3/api-docs.
- If you are getting port already in use error, open terminal and execute '**fuser -k 8080/tcp**' or '**sudo service jenkins stop**'. If you find difficult doing this, go to application.properties and change server.port=8080 to any other port. E.g., server.port=8082
- After completing the hands-on, submit the test.

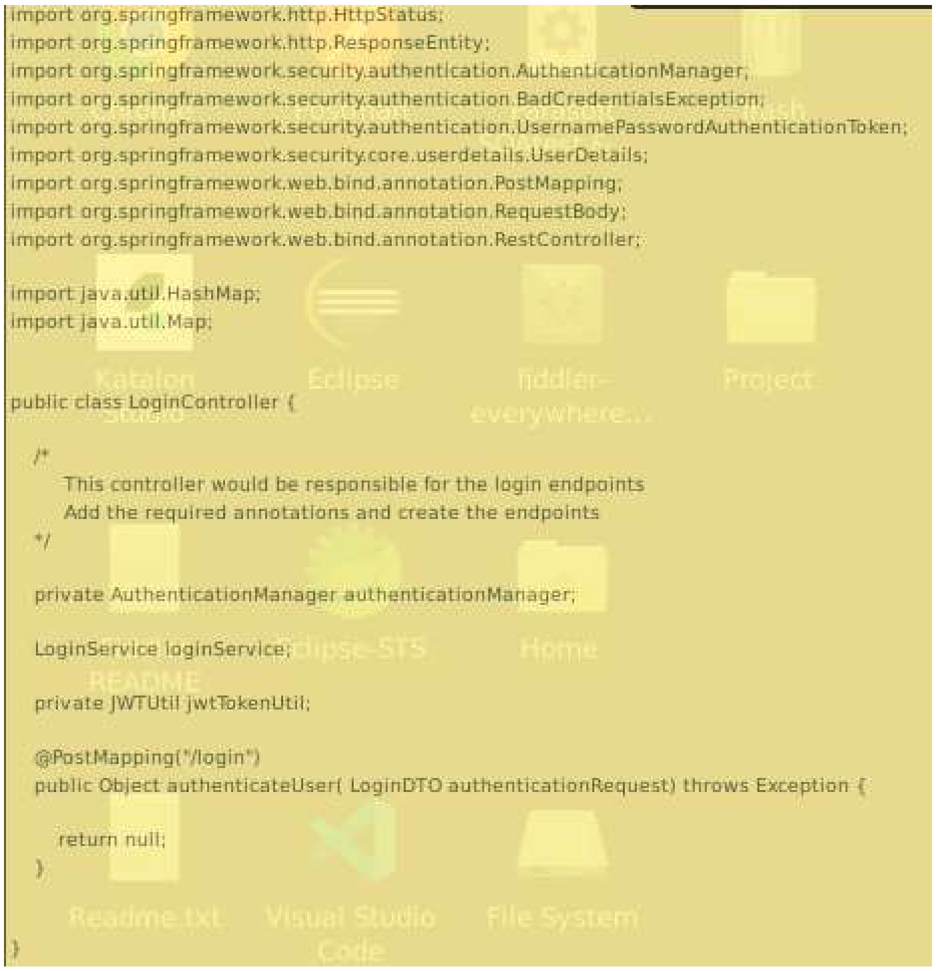
Controllers:

Bidding Controller



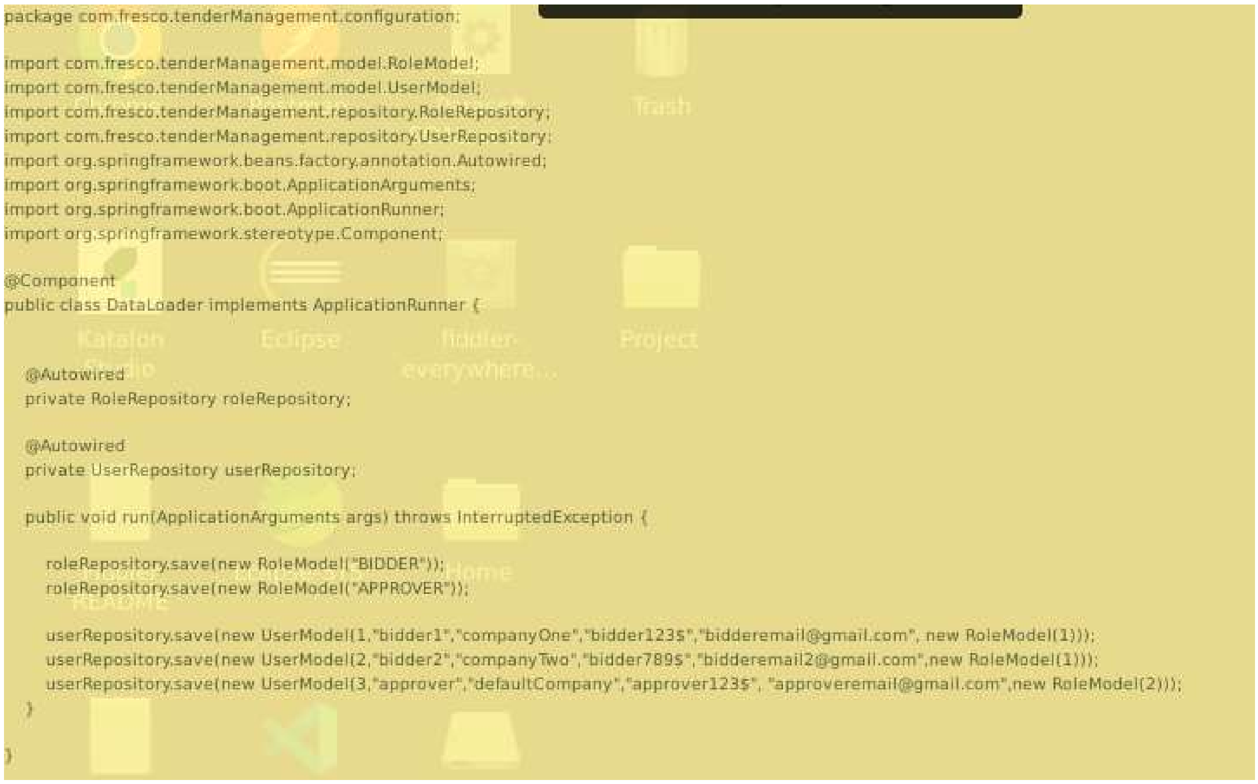
```java
@RequestMapping("/bidding")
public class BiddingController {

  /*
 This controller would be responsible for the BiddingController endpoints
 Add the required annotations and create the endpoints
  */

  private BiddingService biddingService;

  //to create a bidding using biddingModel object
  @PostMapping("/add")
  public ResponseEntity<Object> postBidding(BiddingModel biddingModel){
     return null;
  }

  //to get the bidding which are greater than the given bidAmount
  @GetMapping("/list")
  public ResponseEntity<Object> getBidding( double bidAmount){
     return null;
  }

  //to update the bidding by id as PathVariable and bidding Object
  @PatchMapping("/update/{id}")
  public ResponseEntity<Object> updateBidding( int id,BiddingModel biddingModel){
     return null;
  }

  // to delete the bidding by using id as PathVariable
  @DeleteMapping("/delete/{id}")
  public ResponseEntity<Object> deleteBidding( int id){
     return null;
  }

}
```

Login Controller



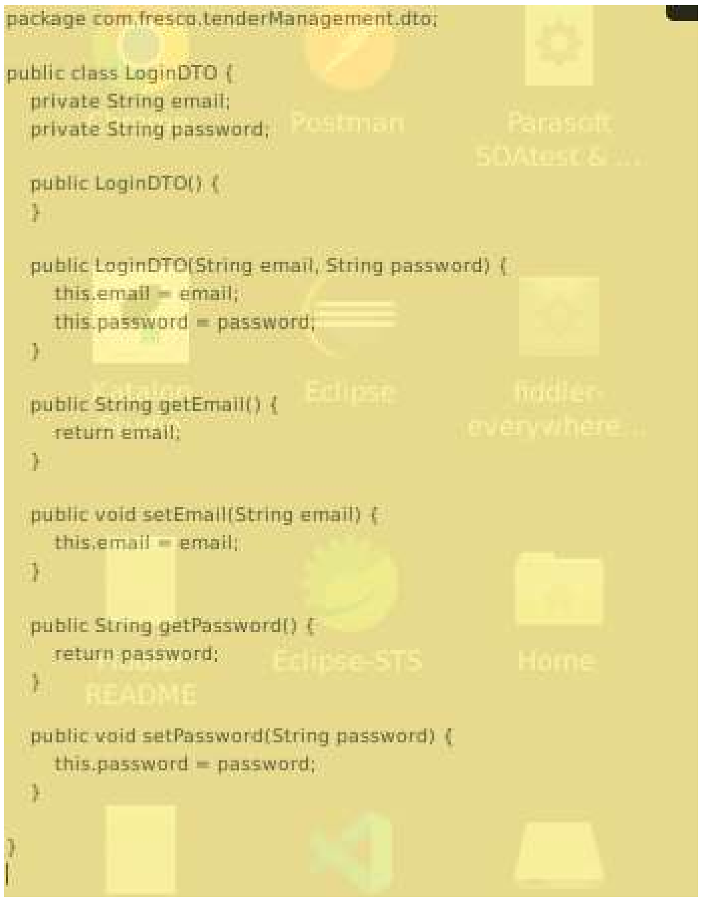
```
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import java.util.HashMap;
import java.util.Map;


public class LoginController {

  /*
     This controller would be responsible for the login endpoints
     Add the required annotations and create the endpoints
  */

  private AuthenticationManager authenticationManager;

  LoginService loginService;

  private JWTUtil jwtTokenUtil;

  @PostMapping("/login")
  public Object authenticateUser( LoginDTO authenticationRequest) throws Exception {

    return null;
  }

}
```

Tendermanagement.configuration.DataLoader.java

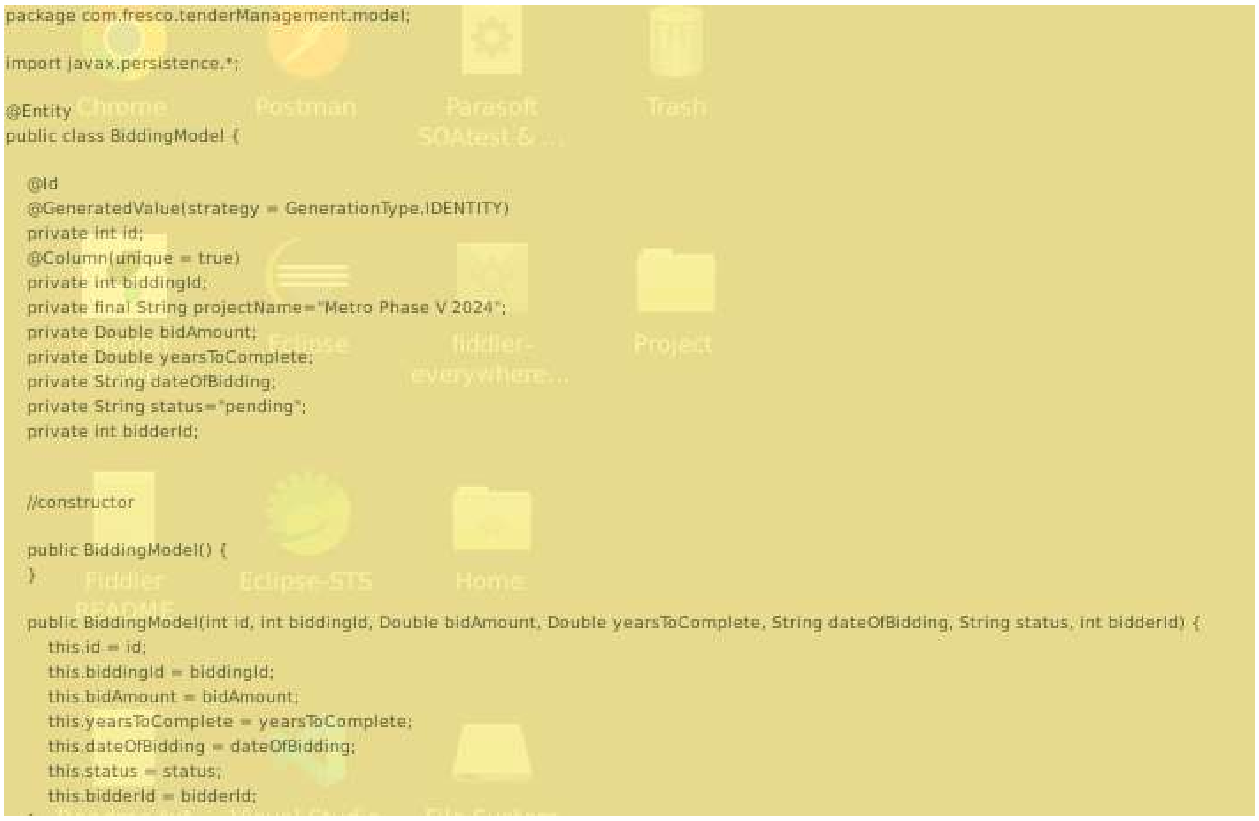


```java
package com.fresco.tenderManagement.configuration;

import com.fresco.tenderManagement.model.RoleModel;
import com.fresco.tenderManagement.model.UserModel;
import com.fresco.tenderManagement.repository.RoleRepository;
import com.fresco.tenderManagement.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.ApplicationArguments;
import org.springframework.boot.ApplicationRunner;
import org.springframework.stereotype.Component;

@Component
public class DataLoader implements ApplicationRunner {

  @Autowired
  private RoleRepository roleRepository;

  @Autowired
  private UserRepository userRepository;

  public void run(ApplicationArguments args) throws InterruptedException {

    roleRepository.save(new RoleModel("BIDDER"));
    roleRepository.save(new RoleModel("APPROVER"));

    userRepository.save(new UserModel(1,"bidder1","companyOne","bidder123$","bidderemail@gmail.com", new RoleModel(1)));
    userRepository.save(new UserModel(2,"bidder2","companyTwo","bidder789$","bidderemail2@gmail.com",new RoleModel(1)));
    userRepository.save(new UserModel(3,"approver","defaultCompany","approver123$", "approveremail@gmail.com",new RoleModel(2)));
  }

}
```

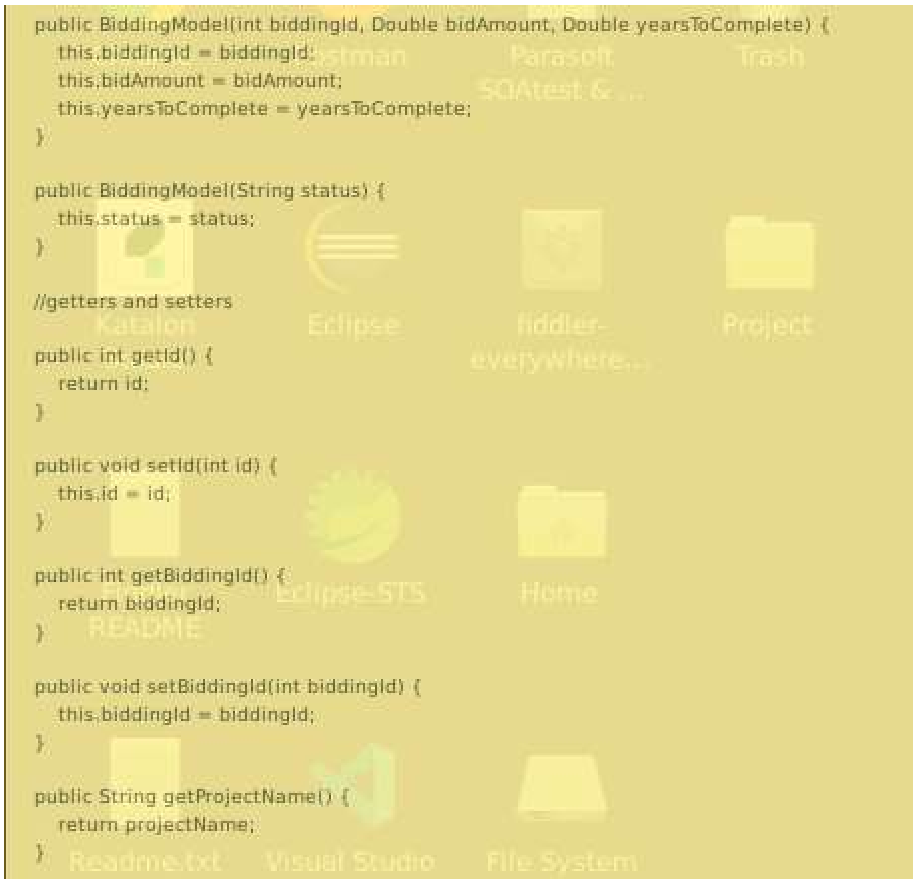
Dto


```
package com.fresco.tenderManagement.dto;

public class LoginDTO {
    private String email;
    private String password;

    public LoginDTO() {
    }

    public LoginDTO(String email, String password) {
        this.email = email;
        this.password = password;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

}
```

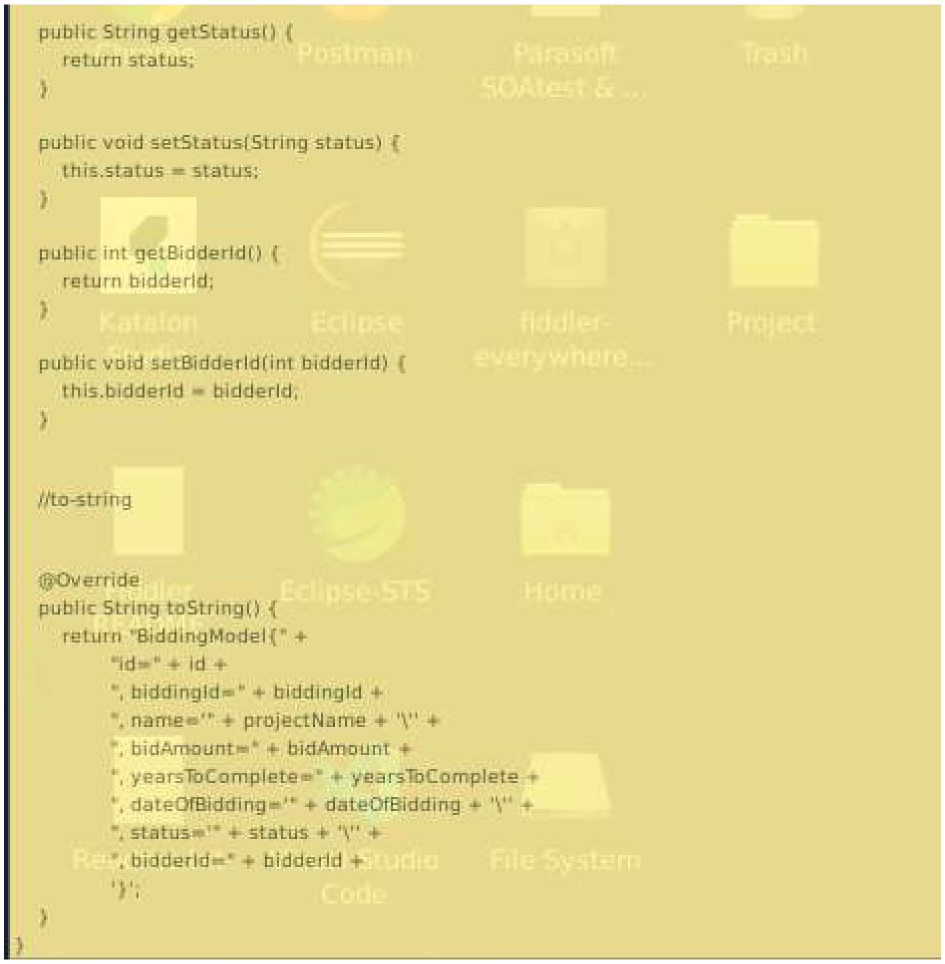
Bidding model



```java
package com.fresco.tenderManagement.model;

import javax.persistence.*;

@Entity
public class BiddingModel {

  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private int id;
  @Column(unique = true)
  private int biddingId;
  private final String projectName="Metro Phase V 2024";
  private Double bidAmount;
  private Double yearsToComplete;
  private String dateOfBidding;
  private String status="pending";
  private int bidderId;


  //constructor

  public BiddingModel() {
  }

  public BiddingModel(int id, int biddingId, Double bidAmount, Double yearsToComplete, String dateOfBidding, String status, int bidderId) {
    this.id = id;
    this.biddingId = biddingId;
    this.bidAmount = bidAmount;
    this.yearsToComplete = yearsToComplete;
    this.dateOfBidding = dateOfBidding;
    this.status = status;
    this.bidderId = bidderId;
```

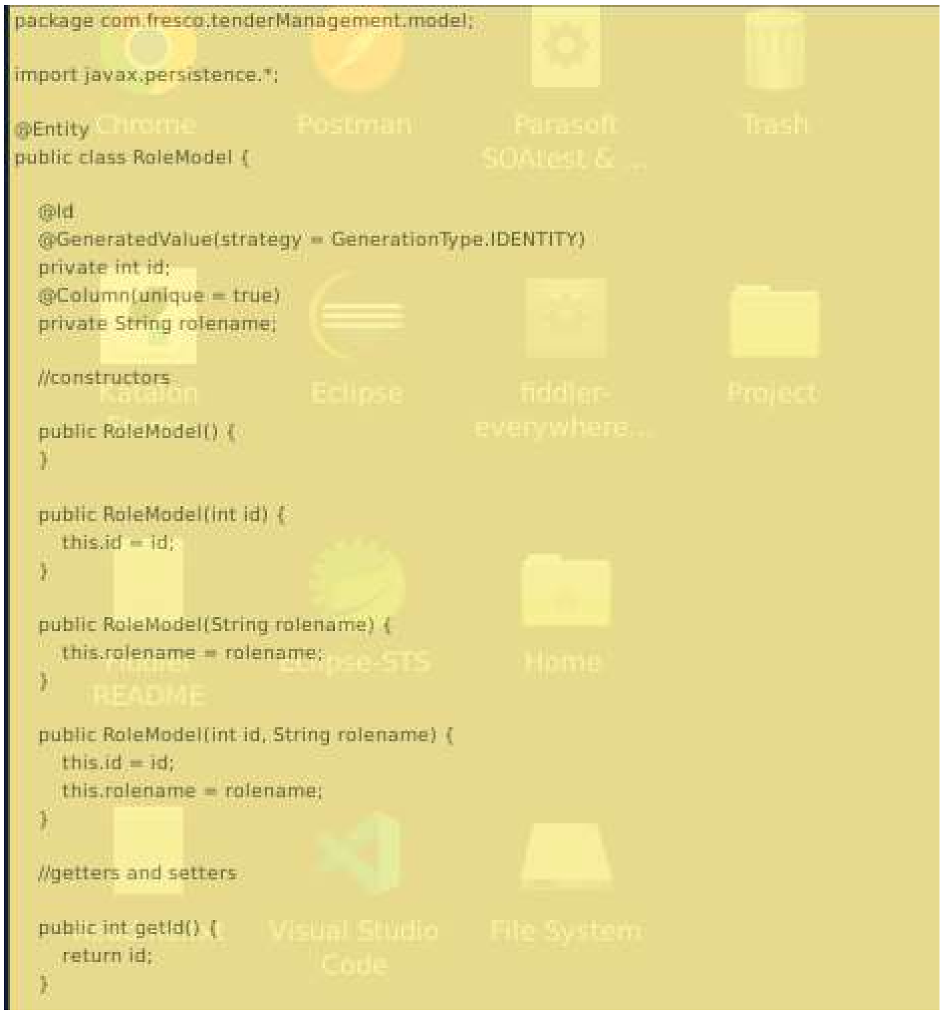


```java
  public BiddingModel(int biddingId, Double bidAmount, Double yearsToComplete) {
    this.biddingId = biddingId;
    this.bidAmount = bidAmount;
    this.yearsToComplete = yearsToComplete;
  }

  public BiddingModel(String status) {
    this.status = status;
  }

  //getters and setters

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public int getBiddingId() {
    return biddingId;
  }

  public void setBiddingId(int biddingId) {
    this.biddingId = biddingId;
  }

  public String getProjectName() {
    return projectName;
  }
```

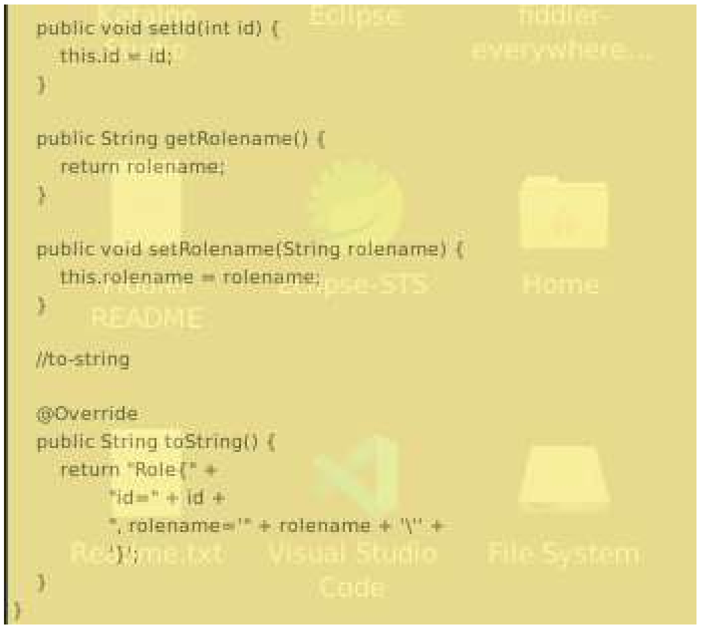


```
    public String getStatus() {
        return status;
    }

    public void setStatus(String status) {
        this.status = status;
    }

    public int getBidderId() {
        return bidderId;
    }

    public void setBidderId(int bidderId) {
        this.bidderId = bidderId;
    }


    //to-string


    @Override
    public String toString() {
        return "BiddingModel{" +
                "id=" + id +
                ", biddingId=" + biddingId +
                ", name='" + projectName + '\'' +
                ", bidAmount=" + bidAmount +
                ", yearsToComplete=" + yearsToComplete +
                ", dateOfBidding='" + dateOfBidding + '\'' +
                ", status='" + status + '\'' +
                ", bidderId=" + bidderId +
                '}';
    }
}
```

Role Model

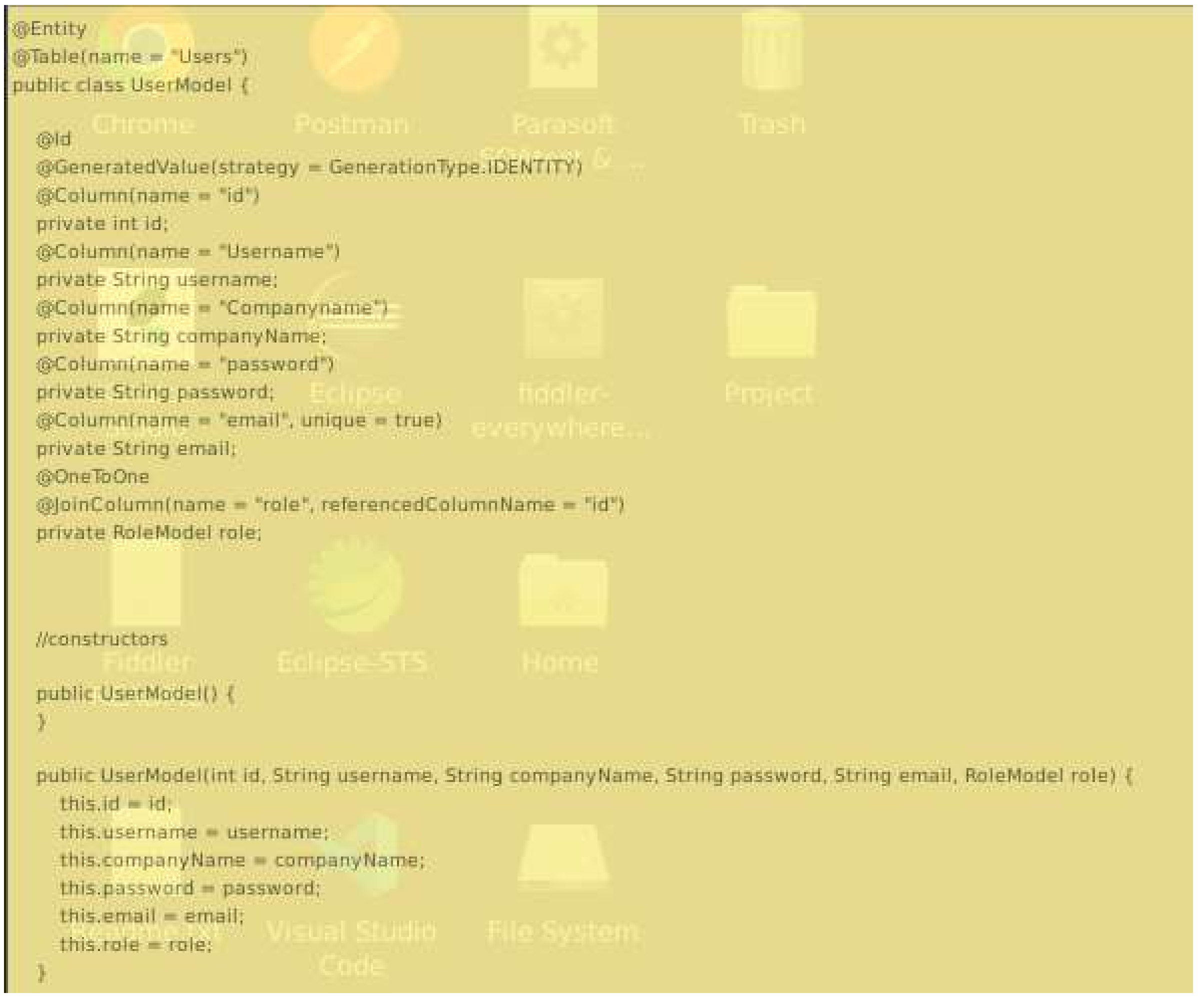


```
package com.fresco.tenderManagement.model;

import javax.persistence.*;

@Entity
public class RoleModel {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    @Column(unique = true)
    private String rolename;

    //constructors

    public RoleModel() {
    }

    public RoleModel(int id) {
        this.id = id;
    }

    public RoleModel(String rolename) {
        this.rolename = rolename;
    }

    public RoleModel(int id, String rolename) {
        this.id = id;
        this.rolename = rolename;
    }

    //getters and setters

    public int getId() {
        return id;
    }
```

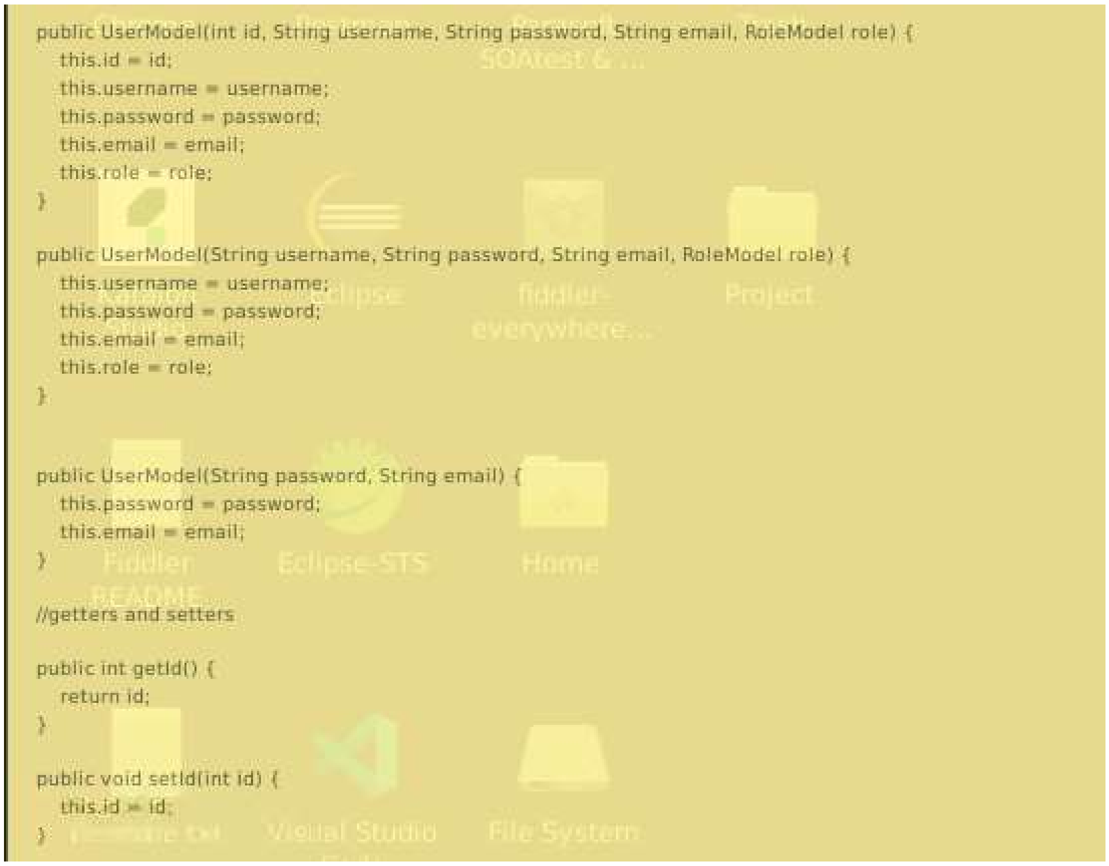


```
    public void setId(int id) {
        this.id = id;
    }

    public String getRolename() {
        return rolename;
    }

    public void setRolename(String rolename) {
        this.rolename = rolename;
    }

    //to-string

    @Override
    public String toString() {
        return "Role{" +
                "id=" + id +
                ", rolename='" + rolename + '\'' +
                '}';
    }
}
```

UserModel



```
@Entity
@Table(name = "Users")
public class UserModel {

   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "id")
   private int id;
   @Column(name = "Username")
   private String username;
   @Column(name = "Companyname")
   private String companyName;
   @Column(name = "password")
   private String password;
   @Column(name = "email", unique = true)
   private String email;
   @OneToOne
   @JoinColumn(name = "role", referencedColumnName = "id")
   private RoleModel role;


   //constructors

   public UserModel() {
   }

   public UserModel(int id, String username, String companyName, String password, String email, RoleModel role) {
      this.id = id;
      this.username = username;
      this.companyName = companyName;
      this.password = password;
      this.email = email;
      this.role = role;
   }
```

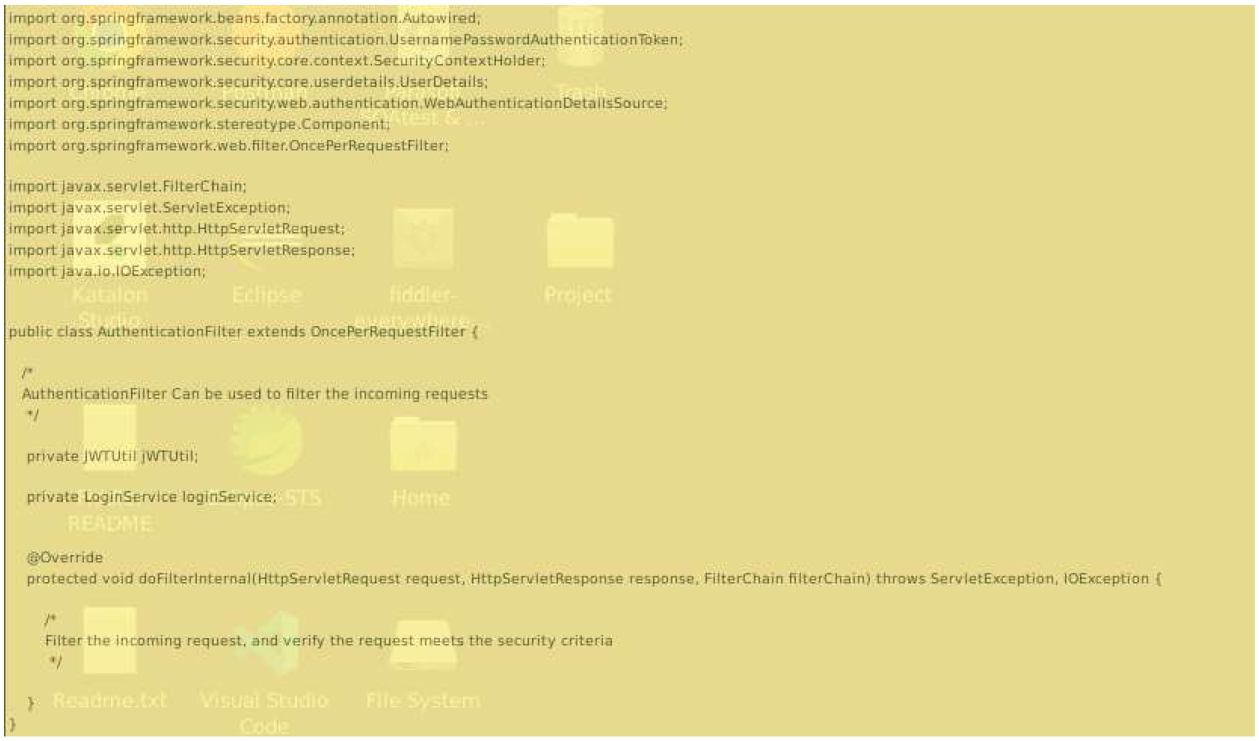


```
public UserModel(int id, String username, String password, String email, RoleModel role) {
    this.id = id;
    this.username = username;
    this.password = password;
    this.email = email;
    this.role = role;
}

public UserModel(String username, String password, String email, RoleModel role) {
    this.username = username;
    this.password = password;
    this.email = email;
    this.role = role;
}


public UserModel(String password, String email) {
    this.password = password;
    this.email = email;
}

//getters and setters

public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}
```

Add getters and setters and toString()

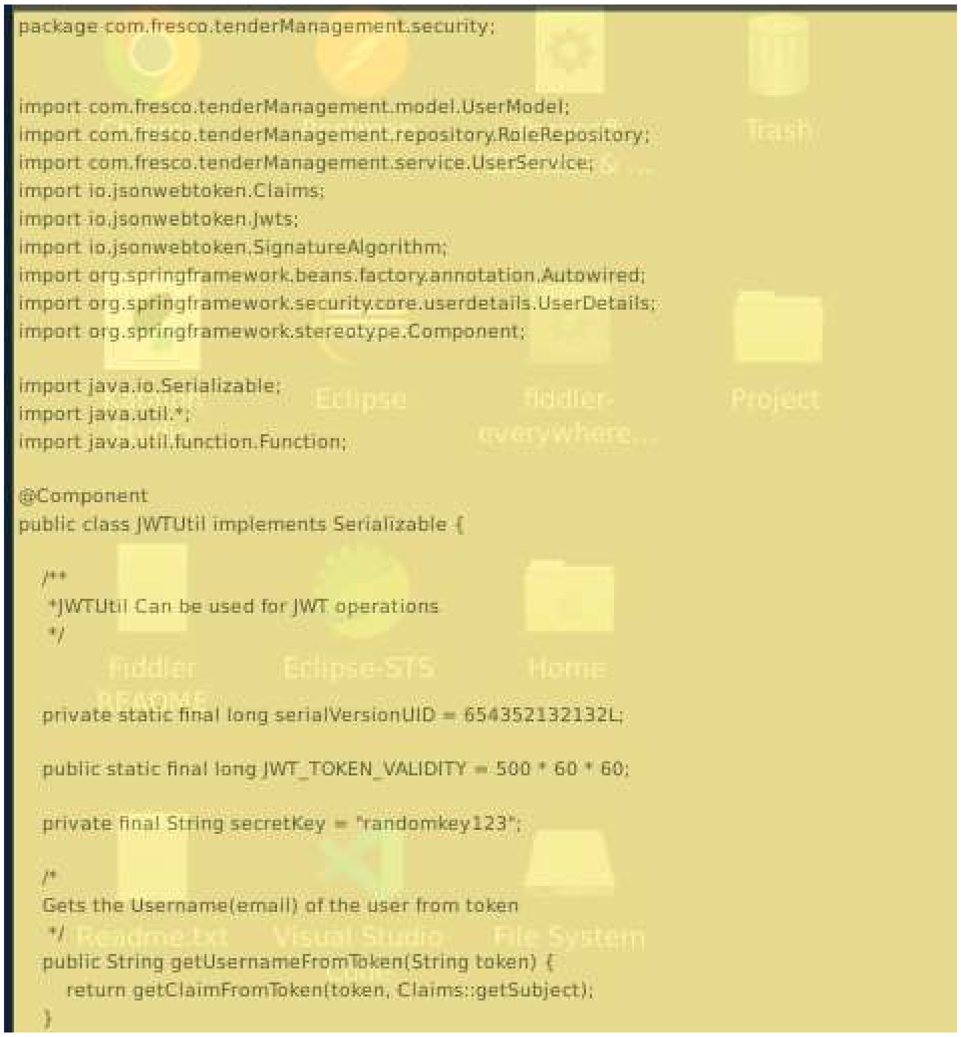
Security

Authentication Filter

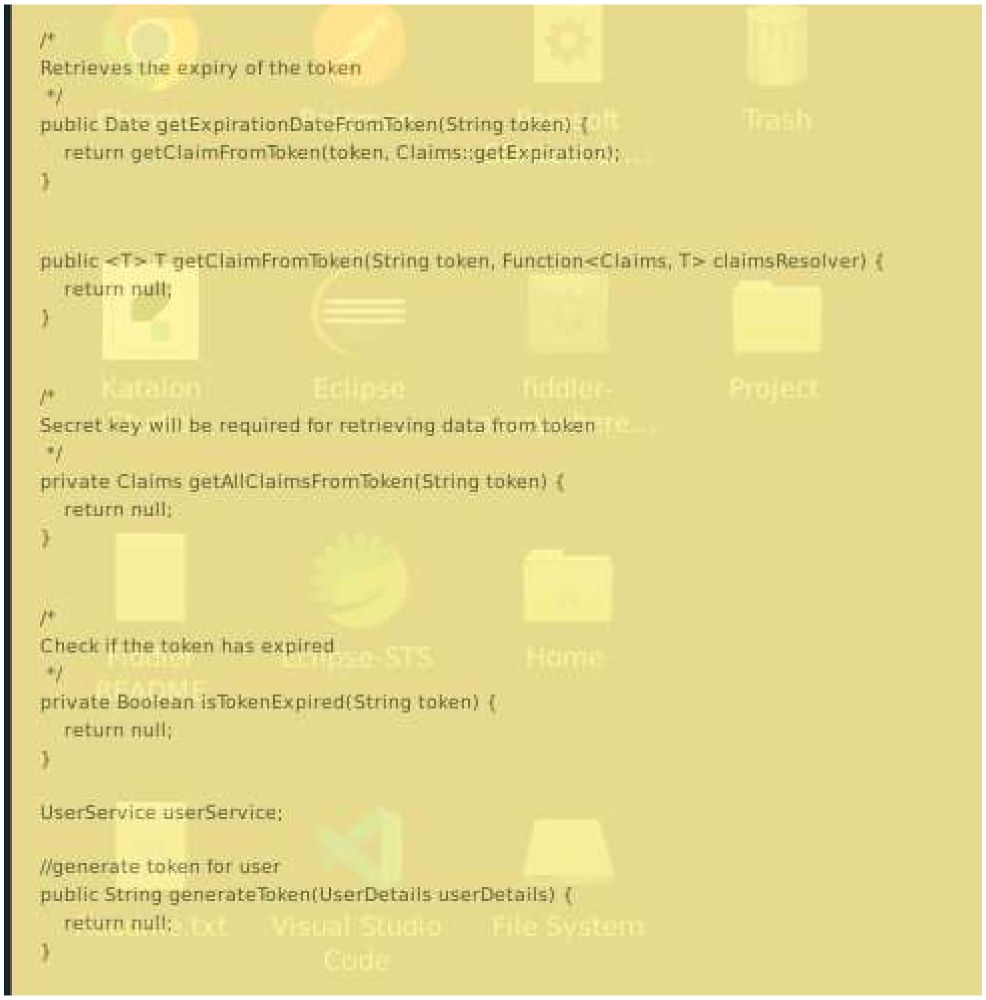


```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;


public class AuthenticationFilter extends OncePerRequestFilter {

  /*
  AuthenticationFilter Can be used to filter the incoming requests
   */

   private JWTUtil jWTUtil;

   private LoginService loginService;


   @Override
   protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {

      /*
      Filter the incoming request, and verify the request meets the security criteria
       */

   }
}
```

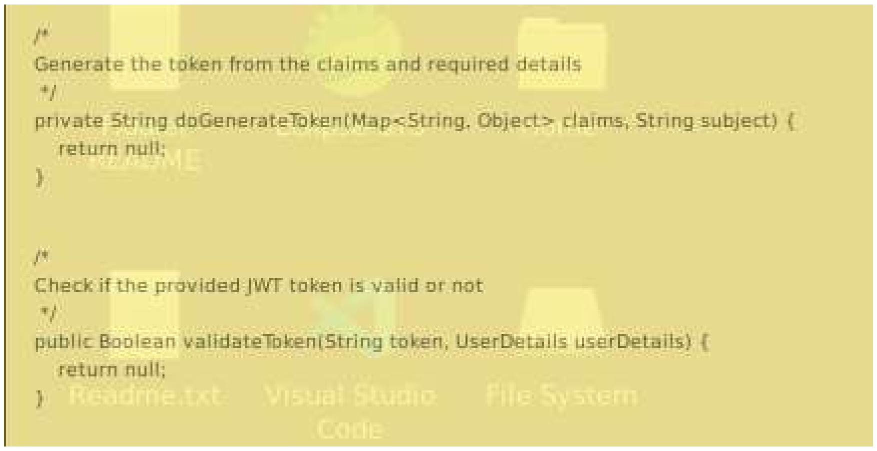
JWT Util



```java
package com.fresco.tenderManagement.security;


import com.fresco.tenderManagement.model.UserModel;
import com.fresco.tenderManagement.repository.RoleRepository;
import com.fresco.tenderManagement.service.UserService;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import java.io.Serializable;
import java.util.*;
import java.util.function.Function;

@Component
public class JWTUtil implements Serializable {

    /**
     *JWTUtil Can be used for JWT operations
     */


    private static final long serialVersionUID = 654352132132L;

    public static final long JWT_TOKEN_VALIDITY = 500 * 60 * 60;

    private final String secretKey = "randomkey123";

    /*
    Gets the Username(email) of the user from token
     */
    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }
```

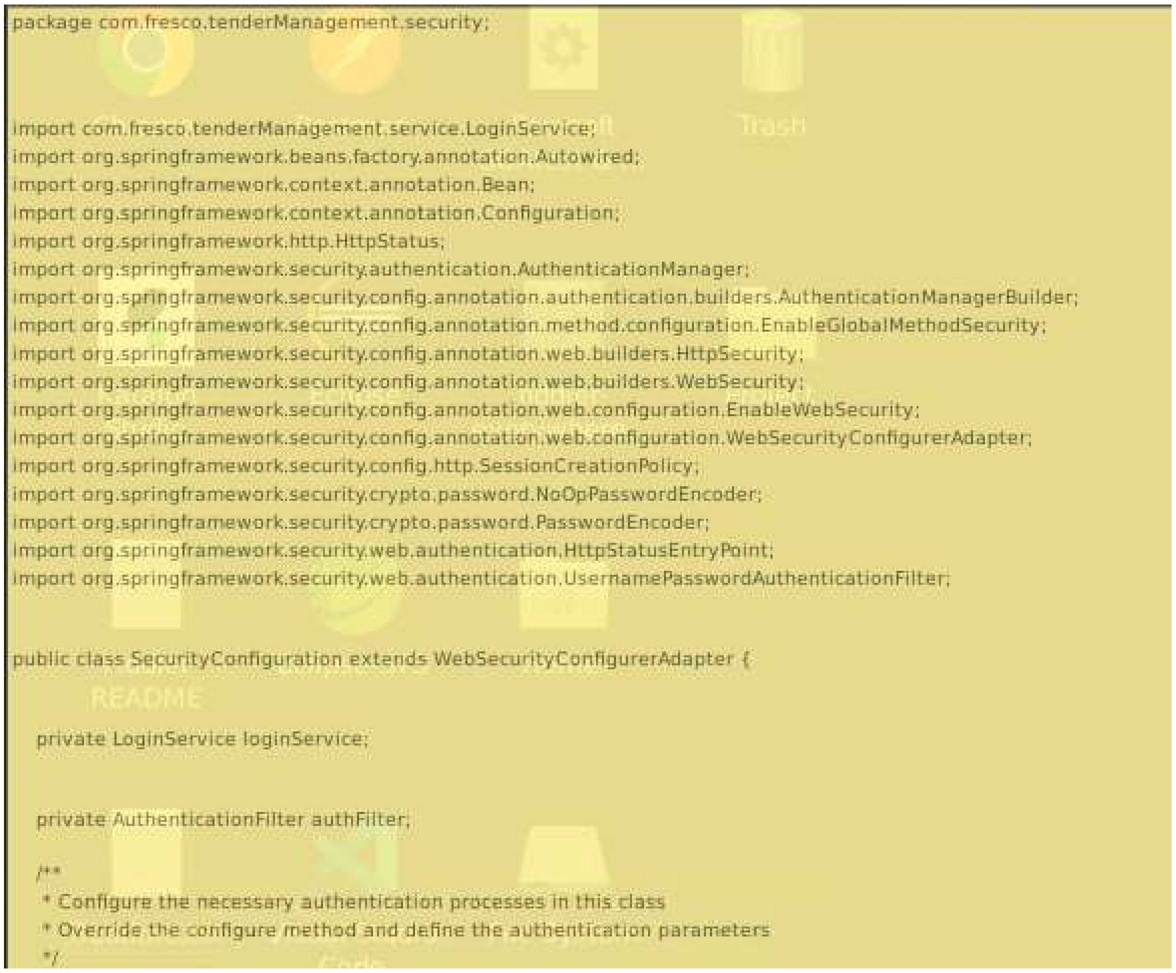


```
/*
Retrieves the expiry of the token
 */
public Date getExpirationDateFromToken(String token) {
    return getClaimFromToken(token, Claims::getExpiration);
}


public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
    return null;
}


/*
Secret key will be required for retrieving data from token
 */
private Claims getAllClaimsFromToken(String token) {
    return null;
}


/*
Check if the token has expired
 */
private Boolean isTokenExpired(String token) {
    return null;
}

UserService userService;

//generate token for user
public String generateToken(UserDetails userDetails) {
    return null;
}
```

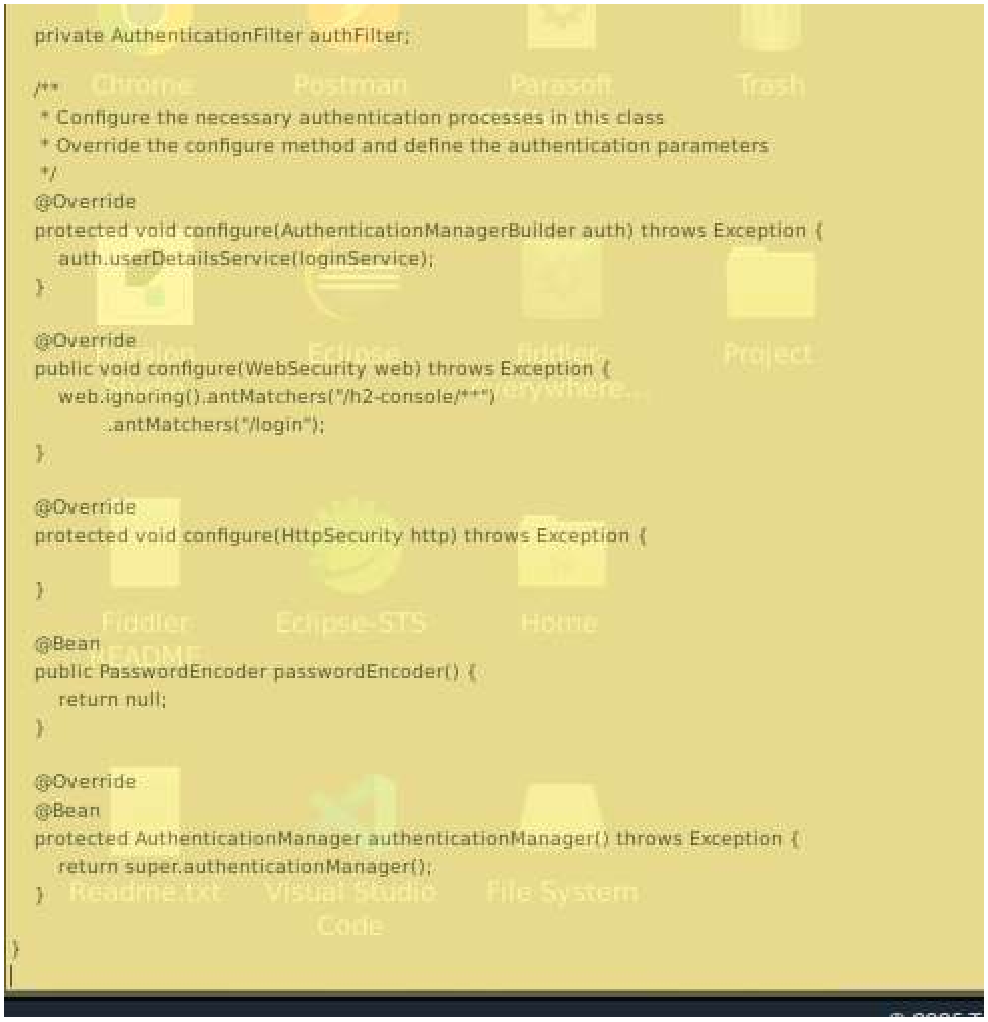


```
/*
Generate the token from the claims and required details
 */
private String doGenerateToken(Map<String, Object> claims, String subject) {
    return null;
}


/*
Check if the provided JWT token is valid or not
 */
public Boolean validateToken(String token, UserDetails userDetails) {
    return null;
}
```

Security Configuration

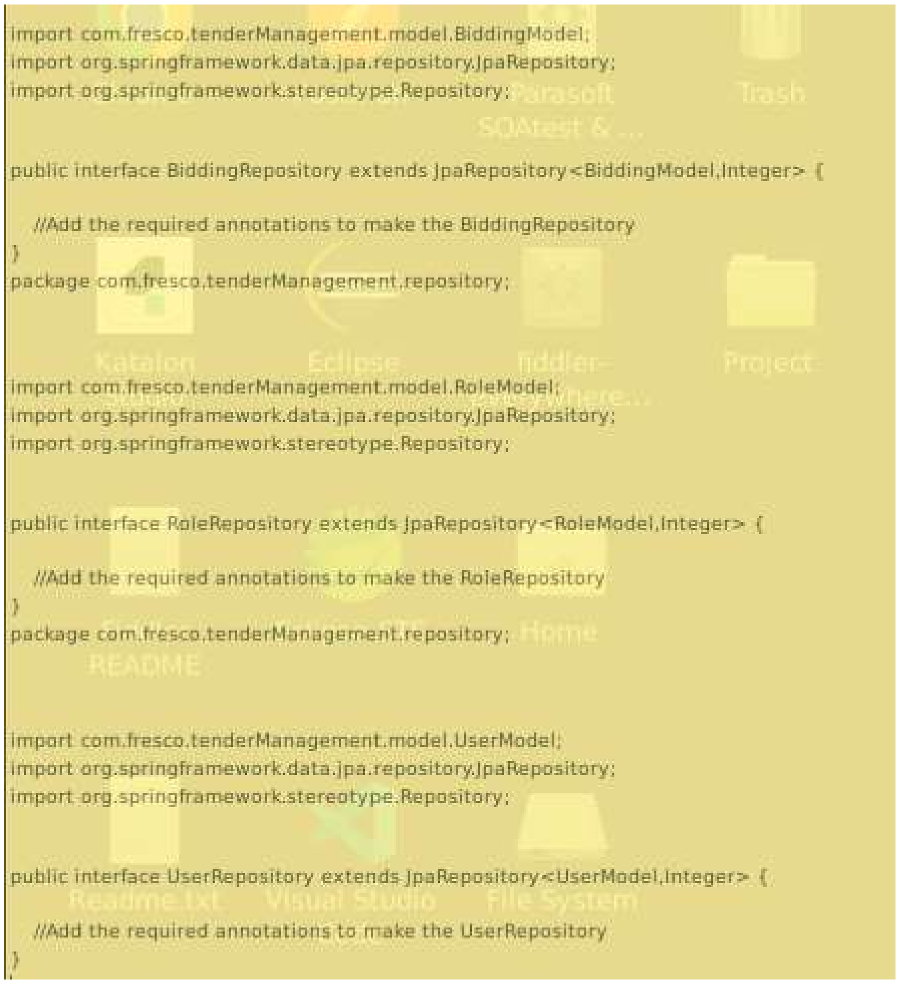


```
package com.fresco.tenderManagement.security;


import com.fresco.tenderManagement.service.LoginService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpStatus;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.builders.WebSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.password.NoOpPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.HttpStatusEntryPoint;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;


public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    private LoginService loginService;


    private AuthenticationFilter authFilter;

    /**
     * Configure the necessary authentication processes in this class
     * Override the configure method and define the authentication parameters
     */
```



```
    private AuthenticationFilter authFilter;

    /**
     * Configure the necessary authentication processes in this class
     * Override the configure method and define the authentication parameters
     */
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(loginService);
    }

    @Override
    public void configure(WebSecurity web) throws Exception {
        web.ignoring().antMatchers("/h2-console/**")
                .antMatchers("/login");
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {

    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return null;
    }

    @Override
    @Bean
    protected AuthenticationManager authenticationManager() throws Exception {
        return super.authenticationManager();
    }

}
```

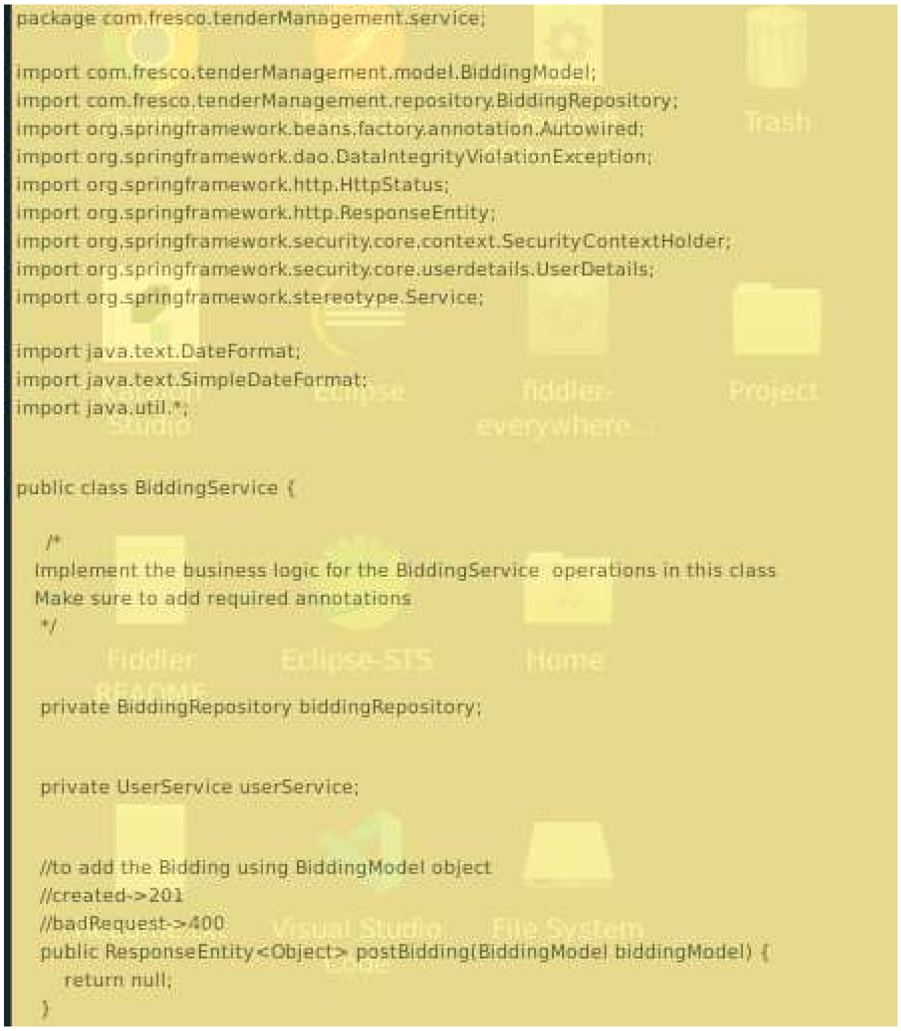
Bidding,User,Role Repos



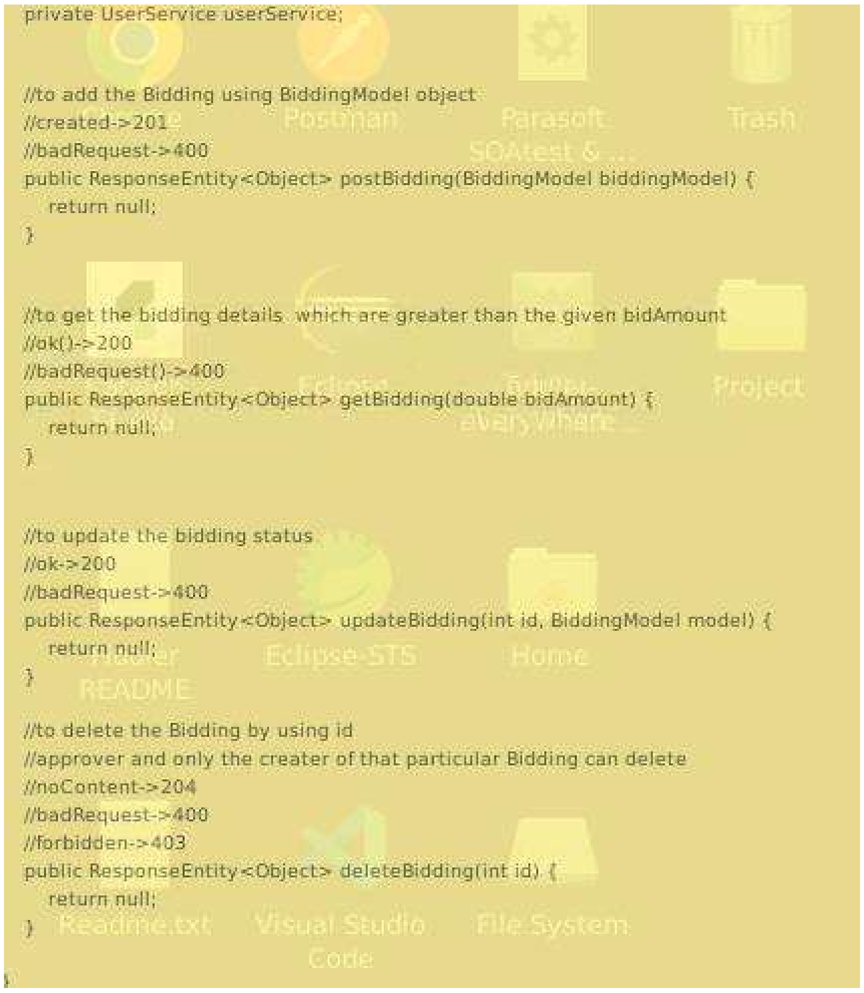
```
import com.fresco.tenderManagement.model.BiddingModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

public interface BiddingRepository extends JpaRepository<BiddingModel,Integer> {

    //Add the required annotations to make the BiddingRepository
}
package com.fresco.tenderManagement.repository;


import com.fresco.tenderManagement.model.RoleModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;


public interface RoleRepository extends JpaRepository<RoleModel,Integer> {

    //Add the required annotations to make the RoleRepository
}
package com.fresco.tenderManagement.repository;


import com.fresco.tenderManagement.model.UserModel;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;


public interface UserRepository extends JpaRepository<UserModel,Integer> {

    //Add the required annotations to make the UserRepository
}
```

Services

Bidding Service

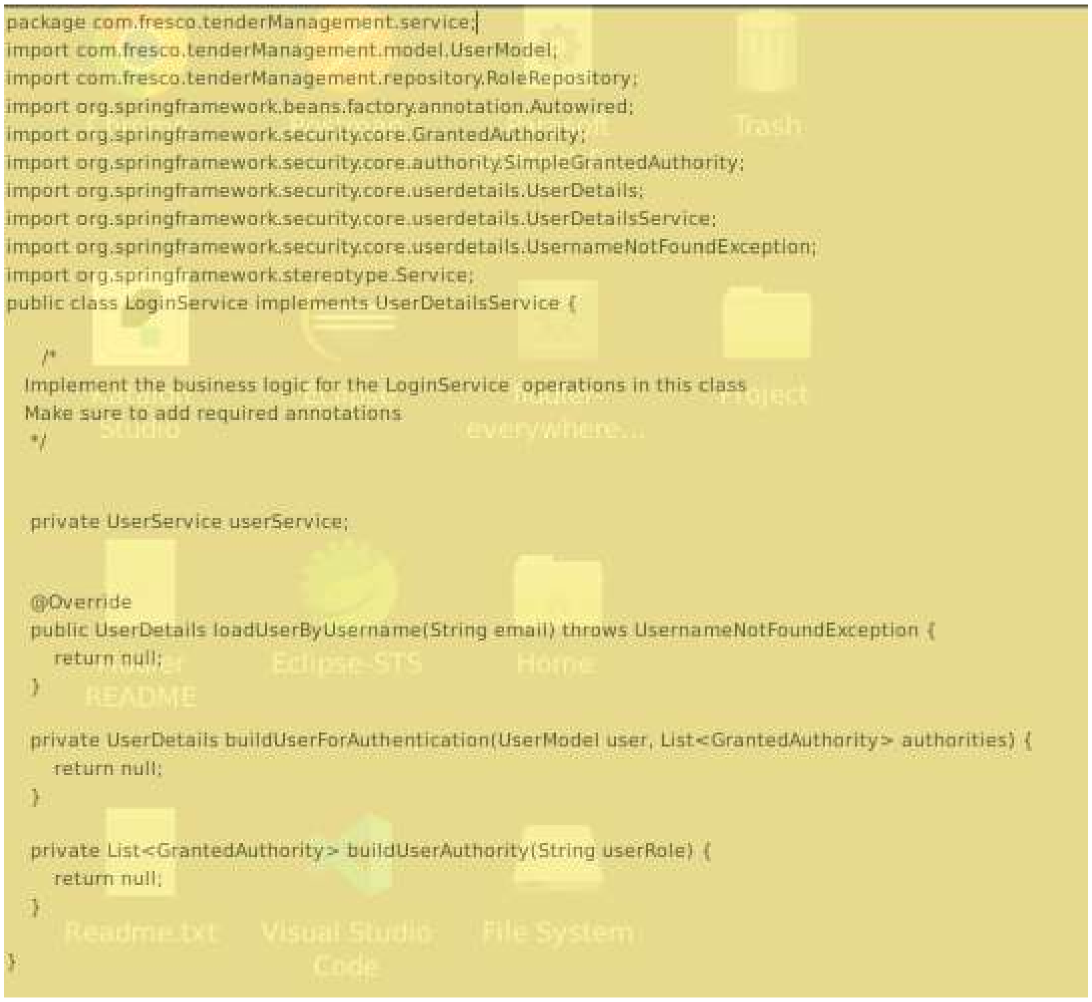


```java
package com.fresco.tenderManagement.service;

import com.fresco.tenderManagement.model.BiddingModel;
import com.fresco.tenderManagement.repository.BiddingRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataIntegrityViolationException;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Service;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.*;


public class BiddingService {

    /*
   Implement the business logic for the BiddingService  operations in this class
   Make sure to add required annotations
    */


    private BiddingRepository biddingRepository;


    private UserService userService;


    //to add the Bidding using BiddingModel object
    //created->201
    //badRequest->400
    public ResponseEntity<Object> postBidding(BiddingModel biddingModel) {
        return null;
    }
```



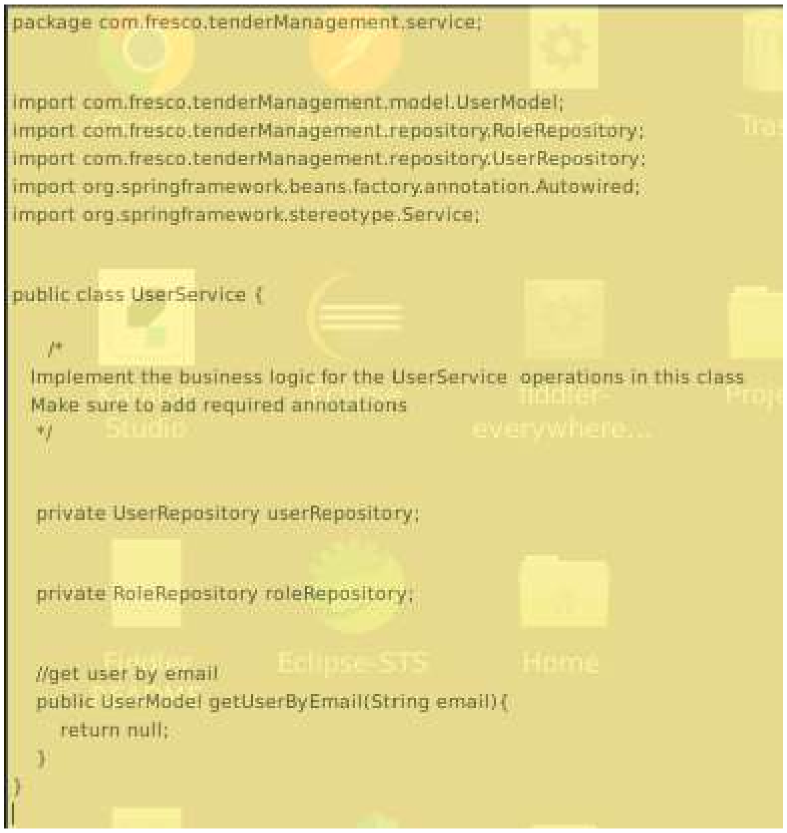
```
  private UserService userService;


  //to add the Bidding using BiddingModel object
  //created->201
  //badRequest->400
  public ResponseEntity<Object> postBidding(BiddingModel biddingModel) {
     return null;
  }


  //to get the bidding details  which are greater than the given bidAmount
  //ok()->200
  //badRequest()->400
  public ResponseEntity<Object> getBidding(double bidAmount) {
     return null;
  }


  //to update the bidding status
  //ok->200
  //badRequest->400
  public ResponseEntity<Object> updateBidding(int id, BiddingModel model) {
     return null;
  }

  //to delete the Bidding by using id
  //approver and only the creater of that particular Bidding can delete
  //noContent->204
  //badRequest->400
  //forbidden->403
  public ResponseEntity<Object> deleteBidding(int id) {
     return null;
  }

}
```

Login Service



```
package com.fresco.tenderManagement.service;
import com.fresco.tenderManagement.model.UserModel;
import com.fresco.tenderManagement.repository.RoleRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
public class LoginService implements UserDetailsService {

   /*
  Implement the business logic for the LoginService  operations in this class
  Make sure to add required annotations
   */


   private UserService userService;


   @Override
   public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
      return null;
   }

   private UserDetails buildUserForAuthentication(UserModel user, List<GrantedAuthority> authorities) {
      return null;
   }

   private List<GrantedAuthority> buildUserAuthority(String userRole) {
      return null;
   }

}
```

User Service

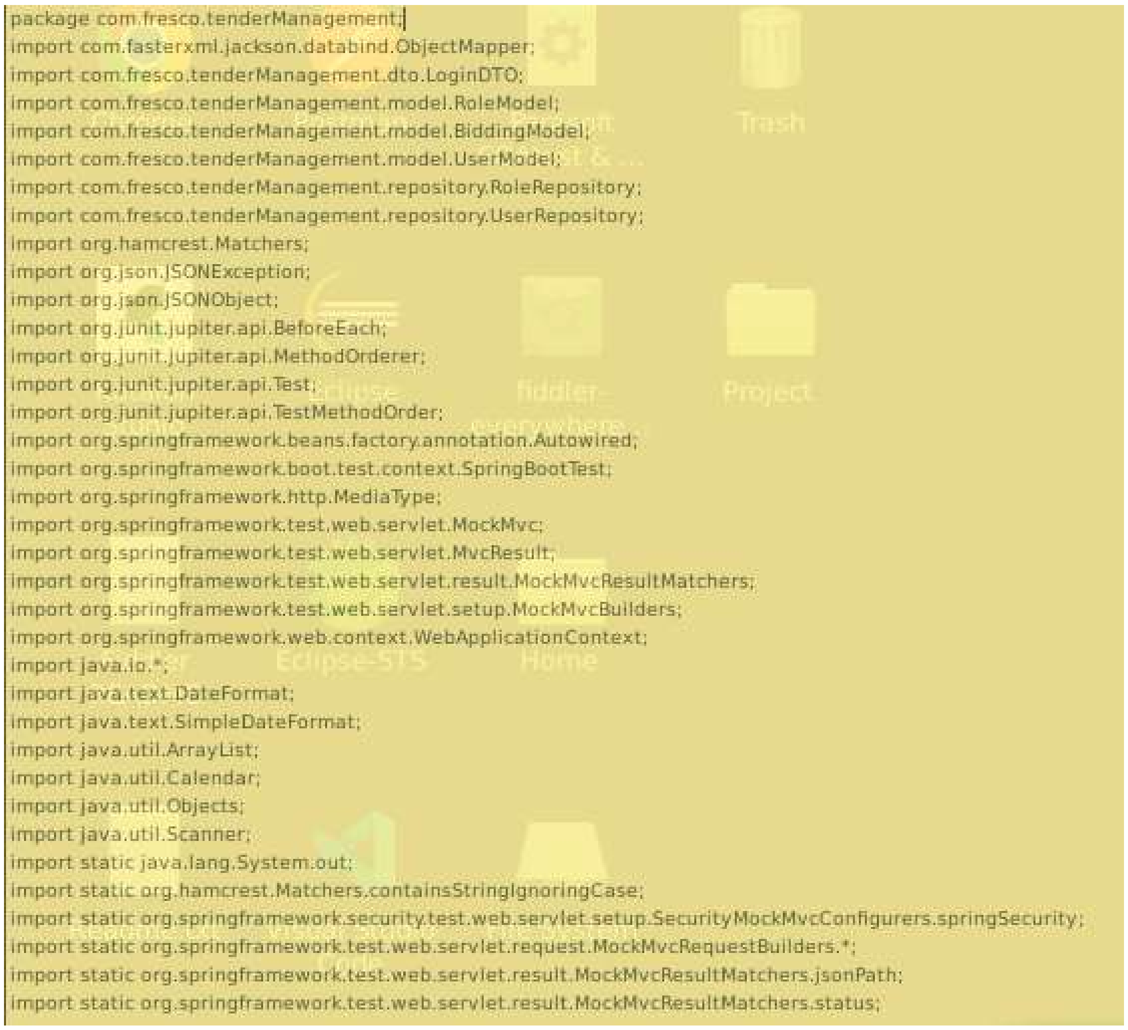


```java
package com.fresco.tenderManagement.service;


import com.fresco.tenderManagement.model.UserModel;
import com.fresco.tenderManagement.repository.RoleRepository;
import com.fresco.tenderManagement.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;


public class UserService {

    /*
   Implement the business logic for the UserService  operations in this class
   Make sure to add required annotations
   */


    private UserRepository userRepository;


    private RoleRepository roleRepository;


    //get user by email
    public UserModel getUserByEmail(String email){
       return null;
    }
}
```

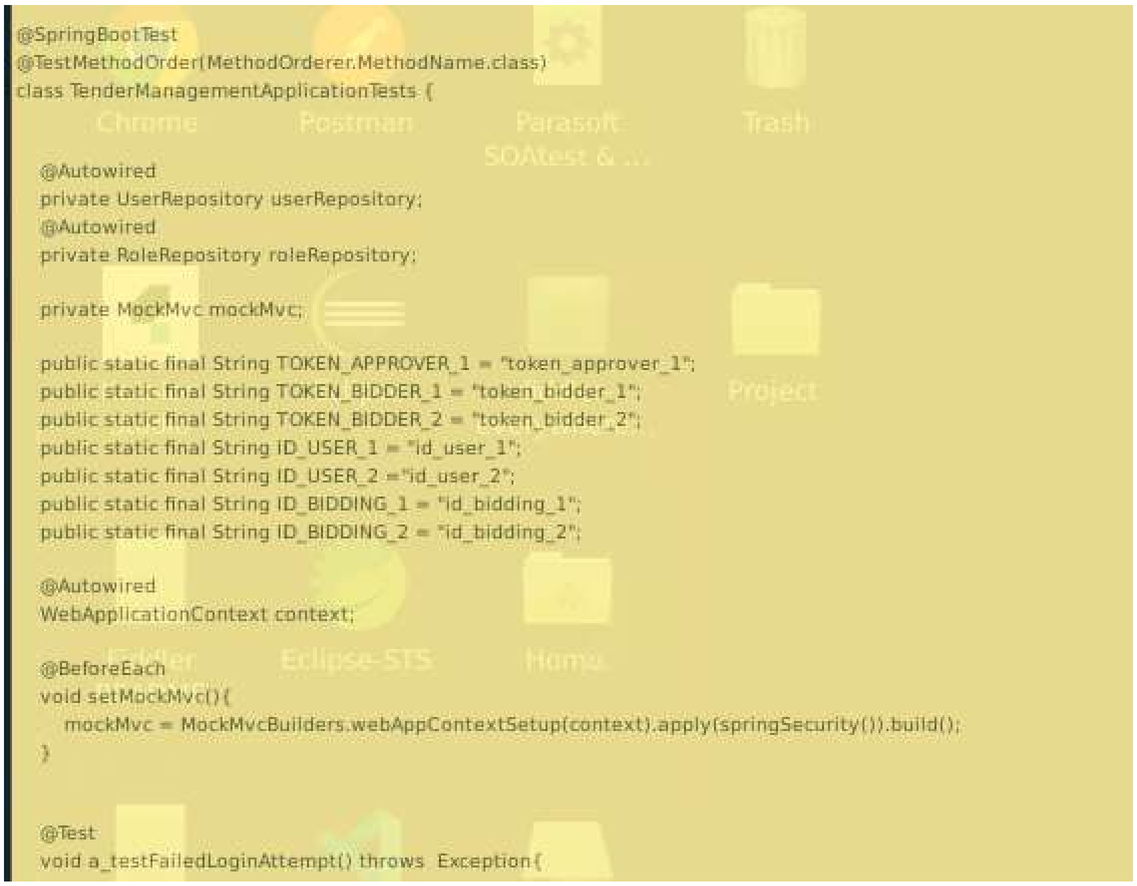
Application.properties

```
spring.jpa.defer-datasource-initialization=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.h2.console.enabled=true
server.port=8080
```

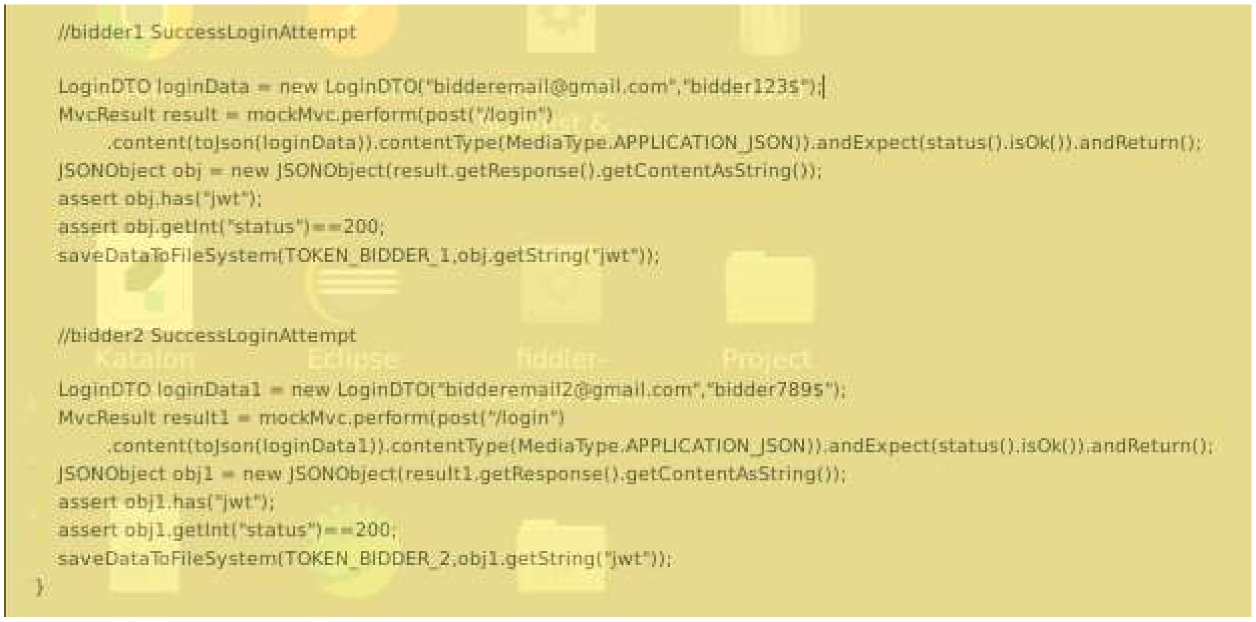
Testcases



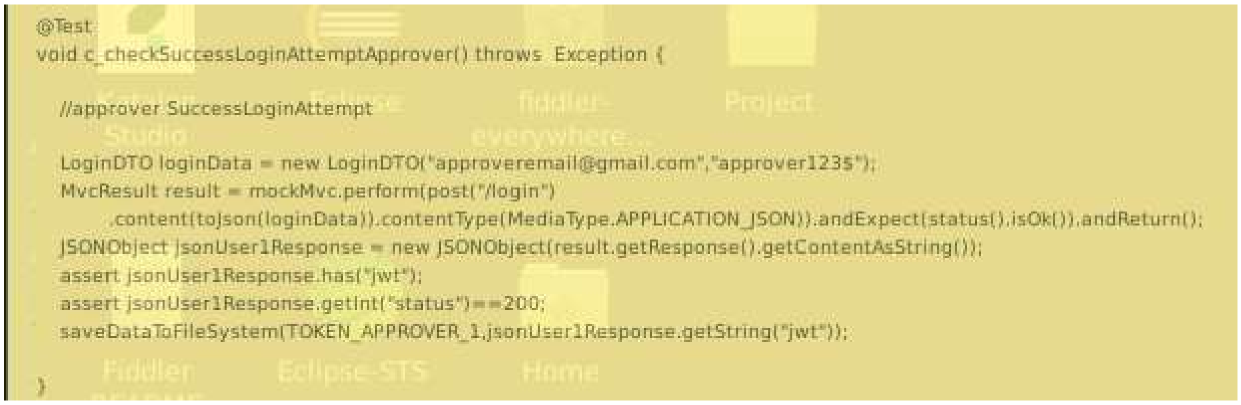
```java
package com.fresco.tenderManagement;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fresco.tenderManagement.dto.LoginDTO;
import com.fresco.tenderManagement.model.RoleModel;
import com.fresco.tenderManagement.model.BiddingModel;
import com.fresco.tenderManagement.model.UserModel;
import com.fresco.tenderManagement.repository.RoleRepository;
import com.fresco.tenderManagement.repository.UserRepository;
import org.hamcrest.Matchers;
import org.json.JSONException;
import org.json.JSONObject;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.MvcResult;
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;
import org.springframework.test.web.servlet.setup.MockMvcBuilders;
import org.springframework.web.context.WebApplicationContext;
import java.io.*;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Objects;
import java.util.Scanner;
import static java.lang.System.out;
import static org.hamcrest.Matchers.containsStringIgnoringCase;
import static org.springframework.security.test.web.servlet.setup.SecurityMockMvcConfigurers.springSecurity;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
```



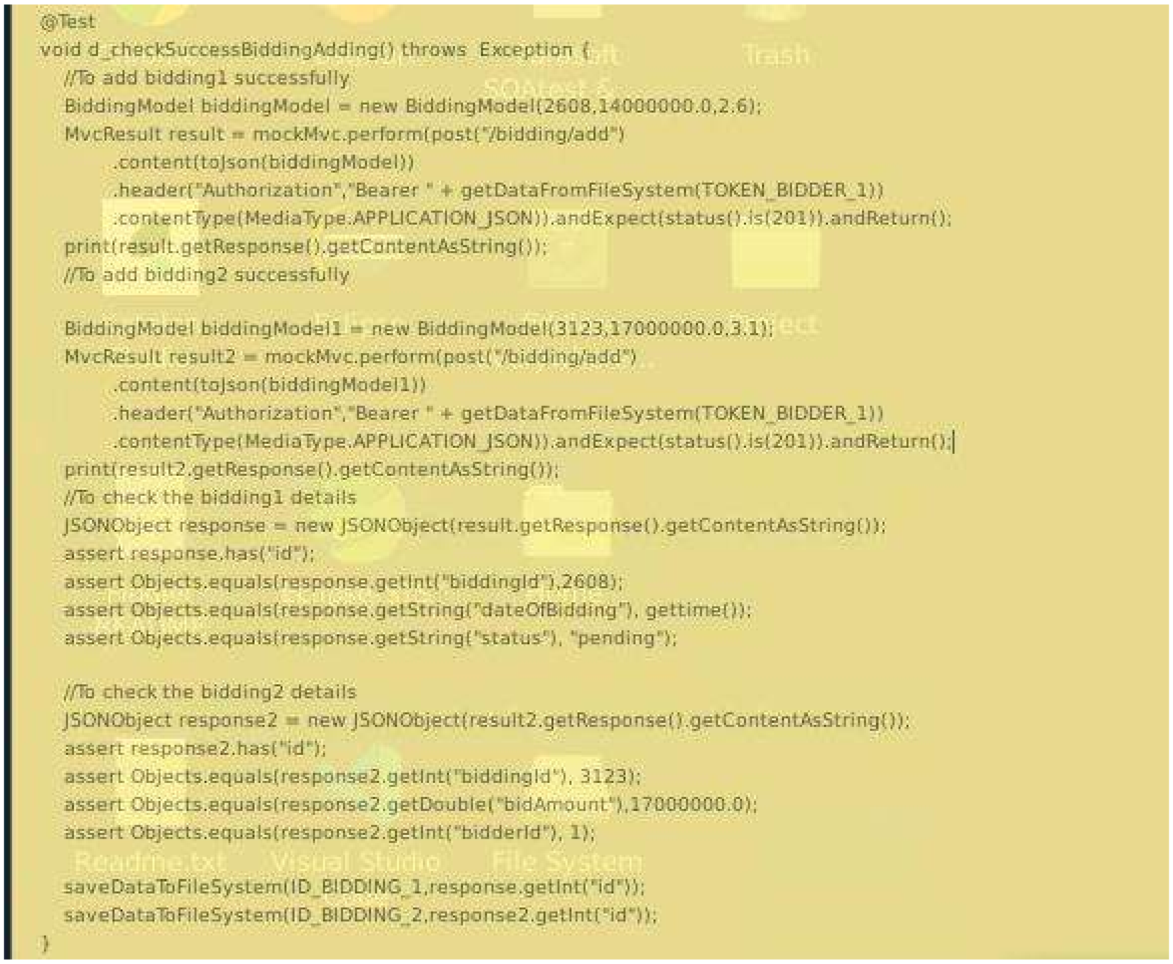
```java
@SpringBootTest
@TestMethodOrder(MethodOrderer.MethodName.class)
class TenderManagementApplicationTests {

   @Autowired
   private UserRepository userRepository;
   @Autowired
   private RoleRepository roleRepository;

   private MockMvc mockMvc;

   public static final String TOKEN_APPROVER_1 = "token_approver_1";
   public static final String TOKEN_BIDDER_1 = "token_bidder_1";
   public static final String TOKEN_BIDDER_2 = "token_bidder_2";
   public static final String ID_USER_1 = "id_user_1";
   public static final String ID_USER_2 ="id_user_2";
   public static final String ID_BIDDING_1 = "id_bidding_1";
   public static final String ID_BIDDING_2 = "id_bidding_2";

   @Autowired
   WebApplicationContext context;

   @BeforeEach
   void setMockMvc(){
      mockMvc = MockMvcBuilders.webAppContextSetup(context).apply(springSecurity()).build();
   }


   @Test
   void a_testFailedLoginAttempt() throws  Exception{
```



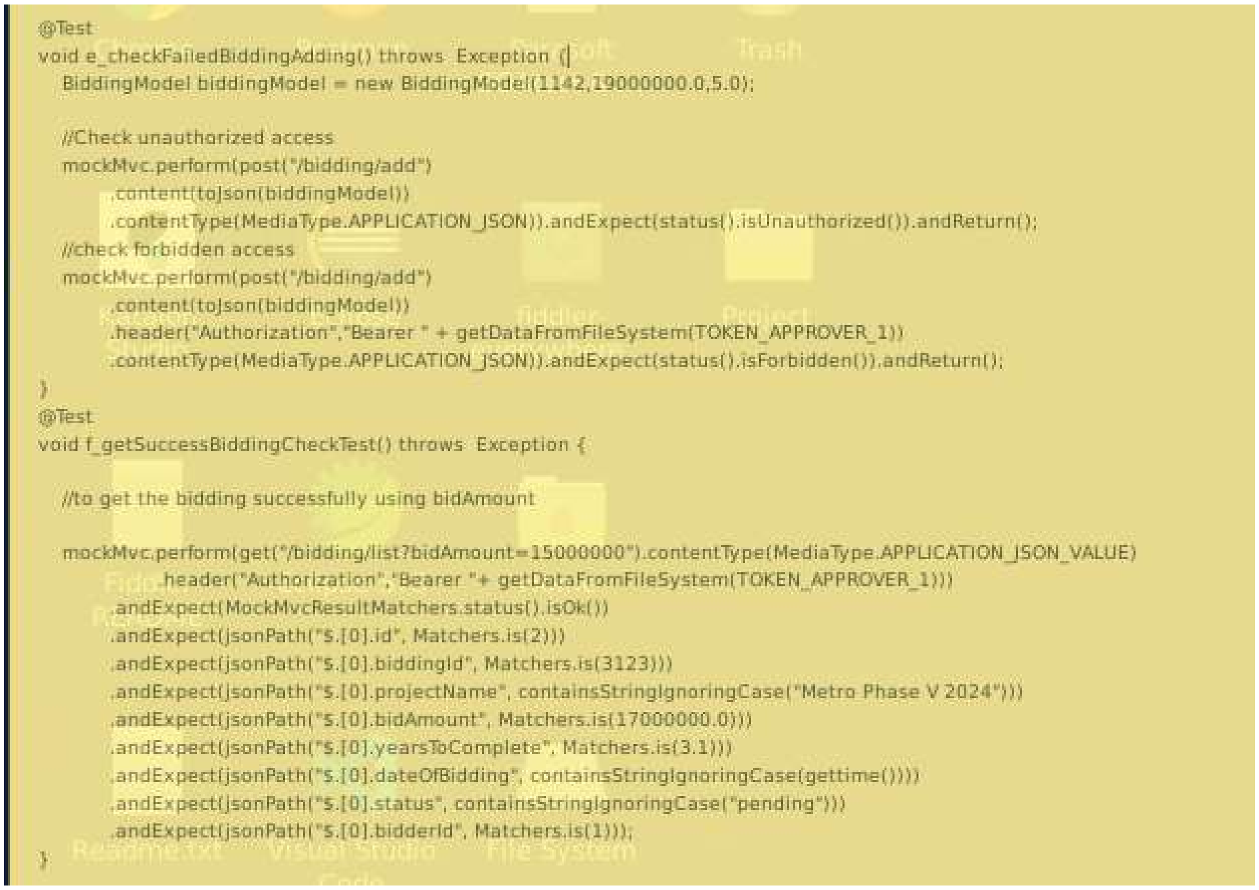
```java
      //bidder1 SuccessLoginAttempt

      LoginDTO loginData = new LoginDTO("bidderemail@gmail.com","bidder123$");
      MvcResult result = mockMvc.perform(post("/login")
            .content(toJson(loginData)).contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk()).andReturn();
      JSONObject obj = new JSONObject(result.getResponse().getContentAsString());
      assert obj.has("jwt");
      assert obj.getInt("status")==200;
      saveDataToFileSystem(TOKEN_BIDDER_1,obj.getString("jwt"));


      //bidder2 SuccessLoginAttempt

      LoginDTO loginData1 = new LoginDTO("bidderemail2@gmail.com","bidder789$");
      MvcResult result1 = mockMvc.perform(post("/login")
            .content(toJson(loginData1)).contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk()).andReturn();
      JSONObject obj1 = new JSONObject(result1.getResponse().getContentAsString());
      assert obj1.has("jwt");
      assert obj1.getInt("status")==200;
      saveDataToFileSystem(TOKEN_BIDDER_2,obj1.getString("jwt"));
   }
```



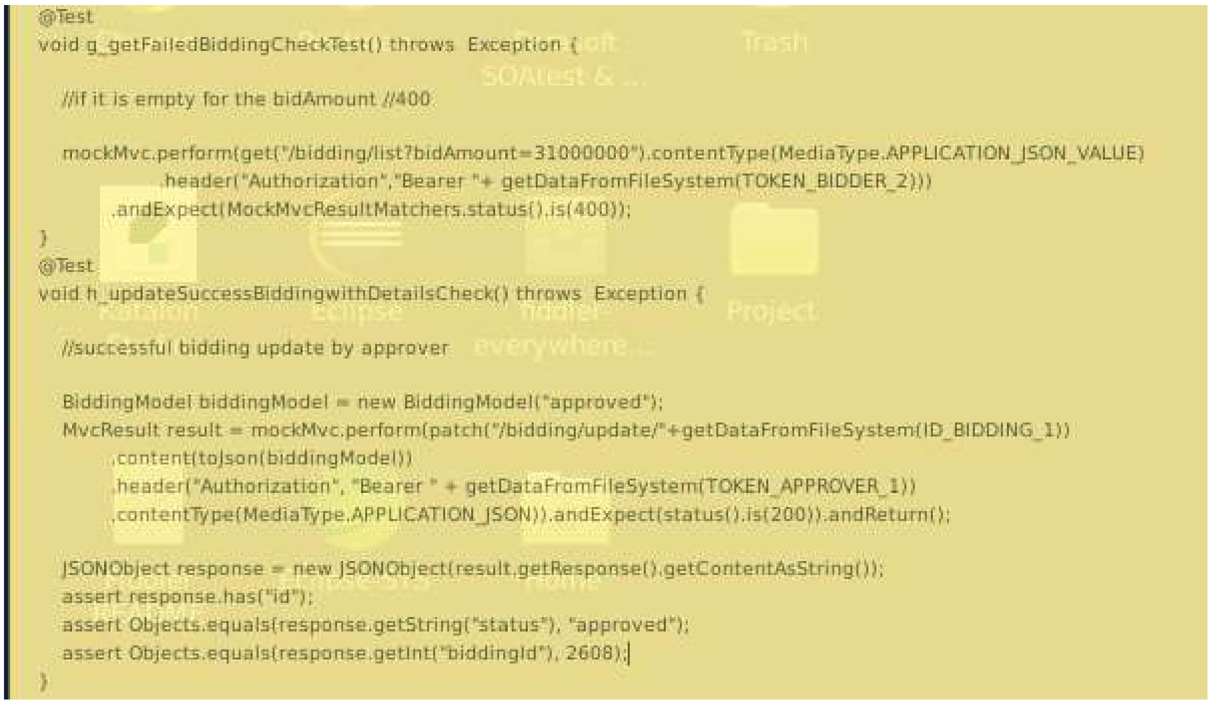
```
@Test
void c_checkSuccessLoginAttemptApprover() throws  Exception {

   //approver SuccessLoginAttempt

   LoginDTO loginData = new LoginDTO("approveremail@gmail.com","approver123$");
   MvcResult result = mockMvc.perform(post("/login")
          .content(toJson(loginData)).contentType(MediaType.APPLICATION_JSON)).andExpect(status().isOk()).andReturn();
   JSONObject jsonUser1Response = new JSONObject(result.getResponse().getContentAsString());
   assert jsonUser1Response.has("jwt");
   assert jsonUser1Response.getInt("status")==200;
   saveDataToFileSystem(TOKEN_APPROVER_1,jsonUser1Response.getString("jwt"));

}
```



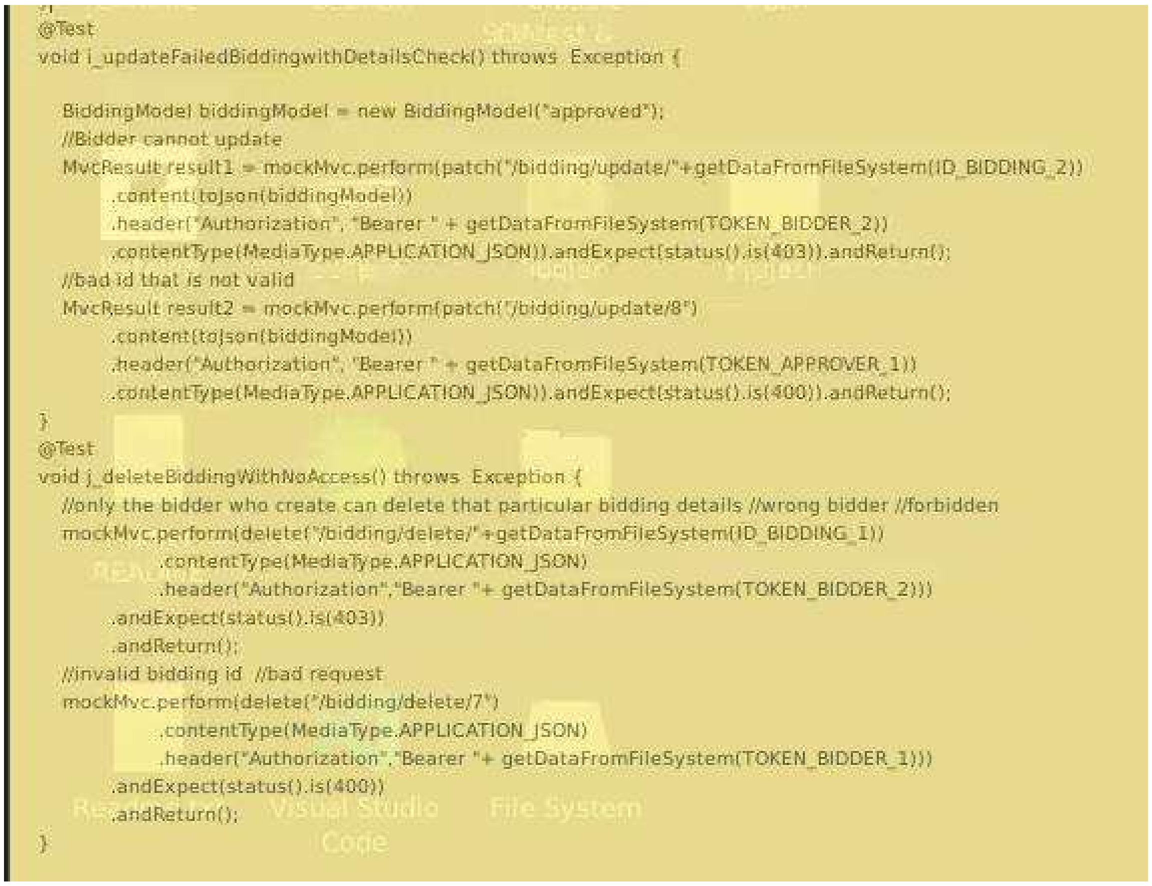
```
@Test
void d_checkSuccessBiddingAdding() throws  Exception {
   //To add bidding1 successfully
   BiddingModel biddingModel = new BiddingModel(2608,14000000.0,2,6);
   MvcResult result = mockMvc.perform(post("/bidding/add")
          .content(toJson(biddingModel))
          .header("Authorization","Bearer " + getDataFromFileSystem(TOKEN_BIDDER_1))
          .contentType(MediaType.APPLICATION_JSON)).andExpect(status().is(201)).andReturn();
   print(result.getResponse().getContentAsString());
   //To add bidding2 successfully

   BiddingModel biddingModel1 = new BiddingModel(3123,17000000.0,3,1);
   MvcResult result2 = mockMvc.perform(post("/bidding/add")
          .content(toJson(biddingModel1))
          .header("Authorization","Bearer " + getDataFromFileSystem(TOKEN_BIDDER_1))
          .contentType(MediaType.APPLICATION_JSON)).andExpect(status().is(201)).andReturn();
   print(result2.getResponse().getContentAsString());
   //To check the bidding1 details
   JSONObject response = new JSONObject(result.getResponse().getContentAsString());
   assert response.has("id");
   assert Objects.equals(response.getInt("biddingId"),2608);
   assert Objects.equals(response.getString("dateOfBidding"), gettime());
   assert Objects.equals(response.getString("status"), "pending");

   //To check the bidding2 details
   JSONObject response2 = new JSONObject(result2.getResponse().getContentAsString());
   assert response2.has("id");
   assert Objects.equals(response2.getInt("biddingId"), 3123);
   assert Objects.equals(response2.getDouble("bidAmount"),17000000.0);
   assert Objects.equals(response2.getInt("bidderId"), 1);

   saveDataToFileSystem(ID_BIDDING_1,response.getInt("id"));
   saveDataToFileSystem(ID_BIDDING_2,response2.getInt("id"));
}
```



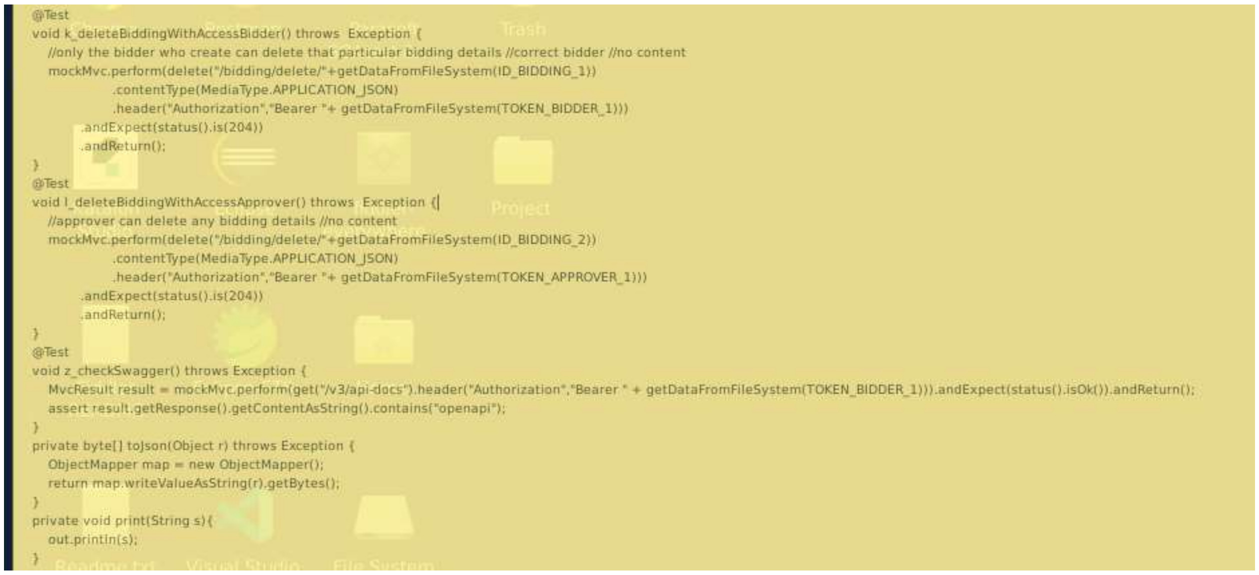
```java
@Test
void e_checkFailedBiddingAdding() throws  Exception {
    BiddingModel biddingModel = new BiddingModel(1142,19000000.0,5.0);

    //Check unauthorized access
    mockMvc.perform(post("/bidding/add")
            .content(toJson(biddingModel))
            .contentType(MediaType.APPLICATION_JSON)).andExpect(status().isUnauthorized()).andReturn();
    //check forbidden access
    mockMvc.perform(post("/bidding/add")
            .content(toJson(biddingModel))
            .header("Authorization","Bearer " + getDataFromFileSystem(TOKEN_APPROVER_1))
            .contentType(MediaType.APPLICATION_JSON)).andExpect(status().isForbidden()).andReturn();
}
@Test
void f_getSuccessBiddingCheckTest() throws  Exception {

    //to get the bidding successfully using bidAmount

    mockMvc.perform(get("/bidding/list?bidAmount=15000000").contentType(MediaType.APPLICATION_JSON_VALUE)
                .header("Authorization","Bearer "+ getDataFromFileSystem(TOKEN_APPROVER_1)))
            .andExpect(MockMvcResultMatchers.status().isOk())
            .andExpect(jsonPath("$.[0].id", Matchers.is(2)))
            .andExpect(jsonPath("$.[0].biddingId", Matchers.is(3123)))
            .andExpect(jsonPath("$.[0].projectName", containsStringIgnoringCase("Metro Phase V 2024")))
            .andExpect(jsonPath("$.[0].bidAmount", Matchers.is(17000000.0)))
            .andExpect(jsonPath("$.[0].yearsToComplete", Matchers.is(3.1)))
            .andExpect(jsonPath("$.[0].dateOfBidding", containsStringIgnoringCase(gettime())))
            .andExpect(jsonPath("$.[0].status", containsStringIgnoringCase("pending")))
            .andExpect(jsonPath("$.[0].bidderId", Matchers.is(1)));
}
```



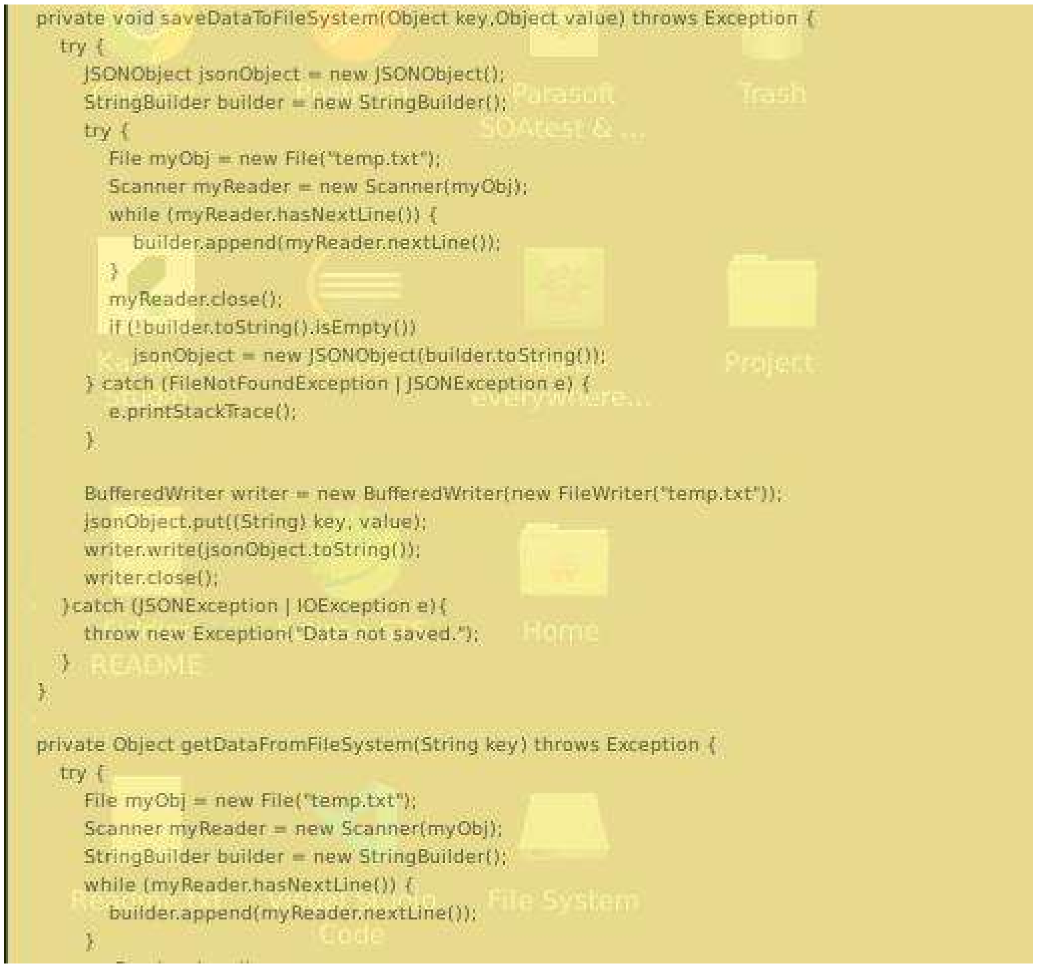
```java
@Test
void g_getFailedBiddingCheckTest() throws  Exception {

    //if it is empty for the bidAmount //400

    mockMvc.perform(get("/bidding/list?bidAmount=31000000").contentType(MediaType.APPLICATION_JSON_VALUE)
                .header("Authorization","Bearer "+ getDataFromFileSystem(TOKEN_BIDDER_2)))
            .andExpect(MockMvcResultMatchers.status().is(400));
}
@Test
void h_updateSuccessBiddingwithDetailsCheck() throws  Exception {

    //successful bidding update by approver

    BiddingModel biddingModel = new BiddingModel("approved");
    MvcResult result = mockMvc.perform(patch("/bidding/update/"+getDataFromFileSystem(ID_BIDDING_1))
            .content(toJson(biddingModel))
            .header("Authorization", "Bearer " + getDataFromFileSystem(TOKEN_APPROVER_1))
            .contentType(MediaType.APPLICATION_JSON)).andExpect(status().is(200)).andReturn();

    JSONObject response = new JSONObject(result.getResponse().getContentAsString());
    assert response.has("id");
    assert Objects.equals(response.getString("status"), "approved");
    assert Objects.equals(response.getInt("biddingId"), 2608);
}
```



```
@Test
void i_updateFailedBiddingwithDetailsCheck() throws  Exception {

    BiddingModel biddingModel = new BiddingModel("approved");
    //Bidder cannot update
    MvcResult result1 = mockMvc.perform(patch("/bidding/update/"+getDataFromFileSystem(ID_BIDDING_2))
            .content(toJson(biddingModel))
            .header("Authorization", "Bearer " + getDataFromFileSystem(TOKEN_BIDDER_2))
            .contentType(MediaType.APPLICATION_JSON)).andExpect(status().is(403)).andReturn();
    //bad id that is not valid
    MvcResult result2 = mockMvc.perform(patch("/bidding/update/8")
            .content(toJson(biddingModel))
            .header("Authorization", "Bearer " + getDataFromFileSystem(TOKEN_APPROVER_1))
            .contentType(MediaType.APPLICATION_JSON)).andExpect(status().is(400)).andReturn();
}
@Test
void j_deleteBiddingWithNoAccess() throws  Exception {
    //only the bidder who create can delete that particular bidding details //wrong bidder //forbidden
    mockMvc.perform(delete("/bidding/delete/"+getDataFromFileSystem(ID_BIDDING_1))
                    .contentType(MediaType.APPLICATION_JSON)
                    .header("Authorization","Bearer "+ getDataFromFileSystem(TOKEN_BIDDER_2)))
            .andExpect(status().is(403))
            .andReturn();
    //invalid bidding id  //bad request
    mockMvc.perform(delete("/bidding/delete/7")
                    .contentType(MediaType.APPLICATION_JSON)
                    .header("Authorization","Bearer "+ getDataFromFileSystem(TOKEN_BIDDER_1)))
            .andExpect(status().is(400))
            .andReturn();
}
```



```java
@Test
void k_deleteBiddingWithAccessBidder() throws  Exception {
   //only the bidder who create can delete that particular bidding details //correct bidder //no content
   mockMvc.perform(delete("/bidding/delete/"+getDataFromFileSystem(ID_BIDDING_1))
                .contentType(MediaType.APPLICATION_JSON)
                .header("Authorization","Bearer "+ getDataFromFileSystem(TOKEN_BIDDER_1)))
         .andExpect(status().is(204))
         .andReturn();
}
@Test
void l_deleteBiddingWithAccessApprover() throws  Exception {
   //approver can delete any bidding details //no content
   mockMvc.perform(delete("/bidding/delete/"+getDataFromFileSystem(ID_BIDDING_2))
                .contentType(MediaType.APPLICATION_JSON)
                .header("Authorization","Bearer "+ getDataFromFileSystem(TOKEN_APPROVER_1)))
         .andExpect(status().is(204))
         .andReturn();
}
@Test
void z_checkSwagger() throws Exception {
   MvcResult result = mockMvc.perform(get("/v3/api-docs").header("Authorization","Bearer " + getDataFromFileSystem(TOKEN_BIDDER_1))).andExpect(status().isOk()).andReturn();
   assert result.getResponse().getContentAsString().contains("openapi");
}
private byte[] toJson(Object r) throws Exception {
   ObjectMapper map = new ObjectMapper();
   return map.writeValueAsString(r).getBytes();
}
private void print(String s){
   out.println(s);
}
```



```java
private void saveDataToFileSystem(Object key,Object value) throws Exception {
   try {
      JSONObject jsonObject = new JSONObject();
      StringBuilder builder = new StringBuilder();
      try {
         File myObj = new File("temp.txt");
         Scanner myReader = new Scanner(myObj);
         while (myReader.hasNextLine()) {
            builder.append(myReader.nextLine());
         }
         myReader.close();
         if (!builder.toString().isEmpty())
            jsonObject = new JSONObject(builder.toString());
      } catch (FileNotFoundException | JSONException e) {
         e.printStackTrace();
      }

      BufferedWriter writer = new BufferedWriter(new FileWriter("temp.txt"));
      jsonObject.put((String) key, value);
      writer.write(jsonObject.toString());
      writer.close();
   }catch (JSONException | IOException e){
      throw new Exception("Data not saved.");
   }
}

private Object getDataFromFileSystem(String key) throws Exception {
   try {
      File myObj = new File("temp.txt");
      Scanner myReader = new Scanner(myObj);
      StringBuilder builder = new StringBuilder();
      while (myReader.hasNextLine()) {
         builder.append(myReader.nextLine());
      }
```

```
            myReader.close();
            JSONObject jsonObject = new JSONObject(builder.toString());
            return jsonObject.get(key);
        } catch (FileNotFoundException | JSONException e) {
            throw new Exception("Data not found. Check authentication and ID generations to make sure data is being produced.");
        }
    }

    public String gettime(){
        String x = String.valueOf(System.currentTimeMillis());
        DateFormat formatter = new SimpleDateFormat("dd/MM/yyyy");
        long milliSeconds= Long.parseLong(x);
        Calendar calendar = Calendar.getInstance();
        calendar.setTimeInMillis(milliSeconds);
        return formatter.format(calendar.getTime());
    }
}
```