Liste des tests du besoins « identification »

1. Tests Under repositories
2. package com.fabrication.agent.repositories;  
     
   import com.fabrication.client.repositories.PersonRepository;  
   import com.fabrication.entities.\*;  
   import com.fabrication.utils.Gender;  
   import com.fabrication.utils.PersonStatus;  
   import com.fabrication.utils.StatusInTreatment;  
   import com.fabrication.utils.StatusTreatmentSystemeList;  
   import org.junit.jupiter.api.DisplayName;  
   import org.junit.jupiter.api.Test;  
   import org.springframework.beans.factory.annotation.Autowired;  
   import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;  
   import org.springframework.data.domain.Page;  
   import org.springframework.data.domain.PageRequest;  
     
   import java.time.Instant;  
   import java.util.Date;  
   import java.util.List;  
   import java.util.Optional;  
     
   import static org.assertj.core.api.Assertions.*assertThat*;  
   import static org.junit.jupiter.api.Assertions.*assertFalse*;  
   import static org.junit.jupiter.api.Assertions.*assertTrue*;  
     
   @DataJpaTest  
   class ReferenceDocumentRepositoryTest {  
    @Autowired  
    ReferenceDocumentRepository<Referencedocument> referenceDocumentRepository;  
     
    @Autowired  
    PersonRepository<Person> personRepository;  
     
     
    @Test  
    void givenIdDocument\_itShouldReturnNullReferenceDocument() {  
    //Given  
    Long id = 985L;  
     
    //When  
    Referencedocument referencedocument = referenceDocumentRepository.findReferenceDocumentById(id);  
     
    //Then  
    *assertThat*(referencedocument).isNull();  
     
    }  
     
    @Test  
    void givenIdDocument\_itShouldReturnReferenceDocument() {  
    //Given  
    Long id = this.saveReferenceDocumentAndReturnRefDocId();  
     
    //When  
    Referencedocument referencedocument = referenceDocumentRepository.findReferenceDocumentById(id);  
     
    //Then  
    *assertThat*(referencedocument).isNotNull();  
     
    }  
     
    @Test  
    @DisplayName("given client id it should return null because client don't exist")  
    void givenClientId\_itShouldReturnNullBecauseClientDoNotExist() {  
    //Given  
    Long id =25685L;  
     
    //When  
    List<Object> object = referenceDocumentRepository.getReferenceDocumentByClientId(id);  
     
    //Then  
    *assertTrue*(object.isEmpty());  
    }  
     
    @Test  
    @DisplayName("given client id it should return list of reference document")  
    void givenClientId\_itShouldReturnAListOfReferenceDocument() {  
    //Given  
    Long id = this.saveReferenceDocumentAndReturnIdClient();  
     
    //When  
    List<Object> object = referenceDocumentRepository.getReferenceDocumentByClientId(id);  
     
    //Then  
    *assertThat*(object.size()).isGreaterThanOrEqualTo(1);  
    }  
     
    @Test  
    @DisplayName("given a null client id it should return empty list")  
    void givenANullClientId\_itShouldReturnEmptyList() {  
    //Given  
    Long id = null;  
     
    //When  
    List<Object> object = referenceDocumentRepository.getReferenceDocumentByClientId(id);  
     
    //Then  
    *assertThat*(object).isEmpty();  
    }  
     
     
    @Test  
    void itShouldReturnAListOfReferenceDocument() {  
    //Given  
    StatusTreatmentSystemeList form = StatusTreatmentSystemeList.*FORM*;  
    StatusTreatmentSystemeList finish = StatusTreatmentSystemeList.*FINISH*;  
    this.saveReferenceDocumentForReseach();  
     
    //When  
    List<Object> object = referenceDocumentRepository.findAll(form,finish);  
     
    //Then  
    *assertThat*(object.size()).isGreaterThanOrEqualTo(1);  
    }  
     
    @Test  
    void itShouldReturnNullForReferenceDocument() {  
    //Given  
    StatusTreatmentSystemeList form = StatusTreatmentSystemeList.*FORM*;  
    StatusTreatmentSystemeList finish = StatusTreatmentSystemeList.*FINISH*;  
     
    //When  
    List<Object> object = referenceDocumentRepository.findAll(form,finish);  
     
    //Then  
    *assertTrue*(object.isEmpty());  
    }  
     
    @Test  
    void itShouldReturnAPageOfReferenceDocument() {  
    //Given  
    StatusTreatmentSystemeList build = StatusTreatmentSystemeList.*BUILD*;  
    this.saveReferenceDocumentForReseach();  
     
    //When  
    Page<Referencedocument> object = referenceDocumentRepository.findAll(  
    build,  
    PageRequest.*of*(  
    0,  
    5  
    )  
    );  
     
    //Then  
    *assertThat*(object.getSize()).isGreaterThanOrEqualTo(1);  
    }  
     
    @Test  
    @DisplayName("given client id it should return null because client don't exist for current reference document")  
    void givenClientId\_itShouldReturnNullBecauseClientDoNotExist1() {  
    //Given  
    Long id =256L;  
     
    //When  
    Optional<Object> cni = referenceDocumentRepository.getCurrentReferenceDocumentByClientId(id,StatusTreatmentSystemeList.*FINISH*);  
     
    //Then  
    *assertFalse*(cni.isPresent());  
    }  
     
    @Test  
    @DisplayName("given client id it should return current reference document")  
    void givenClientId\_itShouldGetCurrentReferenceDocument() {  
    //Given  
    Long id = this.saveReferenceDocumentAndReturnIdClient();  
     
    //When  
    Optional<Object> cni = referenceDocumentRepository.getCurrentReferenceDocumentByClientId(id,StatusTreatmentSystemeList.*FINISH*);  
     
    //Then  
     
    *assertThat*(cni.isPresent()).isTrue();  
    // assertThat(cni.getClient()).isEqualTo(personRepository.findClientById(id));  
    }  
     
     
     
    @Test  
    @DisplayName("given client id it should return null because document don't exist")  
    void givenClientId\_itShouldReturnEmptyOptionalDocumentDoNotExist() {  
    //Given  
    Long id =256L;  
     
    //When  
    Optional<Object> cni = referenceDocumentRepository.findOptionalReferenceDocumentById(id);  
     
    //Then  
    *assertFalse*(cni.isPresent());  
    }  
     
    @Test  
    @DisplayName("given client id it should return optional reference document")  
    void givenClientId\_itShouldGetOptionalReferenceDocument() {  
    //Given  
    Long id = this.saveReferenceDocumentAndReturnRefDocId();  
     
    //When  
    Optional<Object> cni = referenceDocumentRepository.findOptionalReferenceDocumentById(id);  
     
    //Then  
    *assertThat*(cni.isPresent()).isTrue();  
    }  
     
    private Passport getPassportFinish(Long id) {  
    return new Passport(  
    12L,  
    null,  
    "123655",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.*from*(Instant.*now*()),  
    Date.*from*(Instant.*now*()),  
    "address",  
    null,  
    personRepository.findClientById(id),  
    null,  
    StatusTreatmentSystemeList.*FINISH*,  
    StatusInTreatment.*Waiting*,  
    null,  
    "Country",  
    null  
    );  
    }  
     
    private Cni getCniForm(Long id) {  
    return new Cni(  
    11L,  
    null,  
    "1236555",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.*from*(Instant.*now*()),  
    Date.*from*(Instant.*now*()),  
    "address",  
    null,  
    personRepository.findClientById(id),  
    null,  
    StatusTreatmentSystemeList.*FORM*,  
    StatusInTreatment.*Waiting*,  
    null  
    );  
    }  
     
    private Passport getPassportBuild(Long id) {  
    return new Passport(  
    1278L,  
    null,  
    "123685",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.*from*(Instant.*now*()),  
    Date.*from*(Instant.*now*()),  
    "address",  
    null,  
    personRepository.findClientById(id),  
    null,  
    StatusTreatmentSystemeList.*BUILD*,  
    StatusInTreatment.*Waiting*,  
    null,  
    "Country",  
    null  
    );  
    }  
     
    private Passport getPassportEmit(Long id) {  
    return new Passport(  
    5278L,  
    null,  
    "123685",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.*from*(Instant.*now*()),  
    Date.*from*(Instant.*now*()),  
    "address",  
    null,  
    personRepository.findClientById(id),  
    null,  
    StatusTreatmentSystemeList.*BUILD*,  
    StatusInTreatment.*Waiting*,  
    null,  
    "Country",  
    null  
    );  
    }  
     
    private Cni getCni(Long id) {  
    return new Cni(  
    2511L,  
    null,  
    "124365",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.*from*(Instant.*now*()),  
    Date.*from*(Instant.*now*()),  
    "address",  
    null,  
    personRepository.findClientById(id),  
    null,  
    StatusTreatmentSystemeList.*VALIDATE*,  
    StatusInTreatment.*Waiting*,  
    null  
    );  
    }  
     
    private Long saveReferenceDocumentAndReturnIdClient(){  
    personRepository.save(  
    new Client(  
    null,  
    "em22222ail@qd.col",  
    PersonStatus.*ACTIVE*,  
    "012345",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    Long id = personRepository.findClientByEmail("em22222ail@qd.col",PersonStatus.*ACTIVE*).getId();  
    referenceDocumentRepository.save(this.getCniForm(id));  
    referenceDocumentRepository.save(this.getPassportFinish(id));  
    return id;  
    }  
     
    private Long saveReferenceDocumentAndReturnRefDocId(){  
    personRepository.save(  
    new Client(  
    null,  
    "defrgt@qd.col",  
    PersonStatus.*ACTIVE*,  
    "012345",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    Long idClient = personRepository.findClientByEmail("defrgt@qd.col",PersonStatus.*ACTIVE*).getId();  
    Long id = referenceDocumentRepository.save(this.getCniForm(idClient)).getIdDocumentReference();  
    return id;  
    }  
     
    private void saveReferenceDocumentForReseach(){  
    personRepository.save(  
    new Client(  
    null,  
    "em222ail@qd.col",  
    PersonStatus.*CREATE*,  
    "012345",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    Long id = personRepository.findClientByEmail("em222ail@qd.col",PersonStatus.*CREATE*).getId();  
    referenceDocumentRepository.save(this.getCni(id));  
    referenceDocumentRepository.save(this.getPassportBuild(id));  
    }  
   }
3. package com.fabrication.client.repositories;  
     
   import com.fabrication.entities.Agent;  
   import com.fabrication.entities.Client;  
   import com.fabrication.utils.PersonStatus;  
   import org.junit.jupiter.api.\*;  
   import org.springframework.beans.factory.annotation.Autowired;  
   import org.springframework.boot.test.autoconfigure.orm.jpa.DataJpaTest;  
     
   import java.time.Instant;  
   import java.util.ArrayList;  
   import java.util.Date;  
   import java.util.List;  
     
   import static org.assertj.core.api.Assertions.*assertThat*;  
   import static org.junit.jupiter.api.Assertions.*assertFalse*;  
   import static org.junit.jupiter.api.Assertions.*assertTrue*;  
     
     
   @DataJpaTest  
   public class PersonRepositoryTest {  
     
    @Autowired  
    PersonRepository<Client> clientRepository;  
     
    @Autowired  
    PersonRepository<Agent> agentRepository;  
     
    @BeforeEach  
    void setUp(){  
    List<Client> clientList = new ArrayList<>();  
    clientList.add(  
    new Client(  
    null,  
    "email1@gmail.com",  
    PersonStatus.*CREATE*,  
    "553456",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    clientList.add(  
    new Client(  
    null,  
    "email@gmail.com",  
    PersonStatus.*CREATE*,  
    "553486",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    clientList.add(  
    new Client(  
    null,  
    "emai@gmail.com",  
    PersonStatus.*ACTIVE*,  
    "553456",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    clientList.add(  
    new Client(  
    null,  
    "mail1@gmail.com",  
    PersonStatus.*CREATE*,  
    "553456",  
    Date.*from*(Instant.*now*())  
    )  
    );  
    clientRepository.saveAll(clientList);  
     
    List<Agent> agentList = new ArrayList<>();  
    agentList.add(  
    new Agent(  
    null,  
    "email@gmail.com",  
    PersonStatus.*CREATE*,  
    "agentLastName",  
    "agentFirstName",  
    "login",  
    "password"  
    )  
    );  
    agentList.add(  
    new Agent(  
    null,  
    "emai45l@gmail.com",  
    PersonStatus.*CREATE*,  
    "qagentLastName",  
    "agentFirstName",  
    "login1",  
    "password"  
    )  
    );  
    agentRepository.saveAll(agentList);  
     
    }  
     
    @AfterEach  
    void destroyAll(){  
    clientRepository.deleteAll();  
    }  
     
    @Test  
    @Order(1)  
    public void givenEmail\_itShouldReturnClient() {  
     
    //given  
    String email = "email1@gmail.com";  
     
    //when  
    Client client = clientRepository.findClientByEmail(email,PersonStatus.*CREATE* );  
     
    //then  
    assertThatClientsAreEqual(email, client);  
    }  
     
    private void assertThatClientsAreEqual(String email, Client client) {  
    *assertThat*(client.getEmail()).isEqualTo(email);  
    *assertThat*(client.getValidationCode()).isEqualTo("553456");  
    *assertThat*(client.getPersonStatus()).isEqualTo(PersonStatus.*CREATE*);  
    *assertThat*(client.getId()).isNotNull();  
     
    }  
     
    @Test  
    @Order(2)  
    public void givenEmail\_itShouldReturnNullClient() {  
     
    //given  
    String email = "titi@gmail.com";  
     
    //when  
    Client client = clientRepository.findClientByEmail(email,PersonStatus.*CREATE* );  
     
    //then  
    *assertThat*(client).isNull();  
    }  
     
    @Test  
    @Order(3)  
    public void givenEmailAndValidationCode\_itShouldActiveClient() {  
    //given  
    String email = "email1@gmail.com";  
    String code = "553456";  
     
    //when  
    Client client = clientRepository.findClientByEmailAndValidationCode(email, code, PersonStatus.*CREATE*);  
     
    //then  
     
    *assertThat*(client).isNotNull();  
    }  
     
    @Test  
    @Order(3)  
    public void givenEmailAndValidationCode\_itShouldNotActiveClient() {  
    //given  
    String email = "email1@gmail.com";  
    String code = "553406";  
     
    //when  
    Client client = clientRepository.findClientByEmailAndValidationCode(email, code, PersonStatus.*CREATE*);  
     
    //then  
     
    *assertThat*(client).isNull();  
    }  
     
    @Test  
    @Order(4)  
    public void givenEmail\_itShouldReturnListOfEntriesForOneClientWhenItIsCreateOrActive() {  
    //given  
    String email = "emai@gmail.com";  
     
    //when  
    List<Client> clientList = clientRepository.findClientByEmailActiveOrCreate(email, PersonStatus.*CREATE*, PersonStatus.*ACTIVE*);  
     
    //then  
    *assertFalse*(clientList.isEmpty());  
    }  
     
    @Test  
    @Order(5)  
    public void givenEmail\_itShouldReturnNullListOfEntriesForOneClientWhenItIsCreateOrActive() {  
    //given  
    String email = "emaitoto@gmail.com";  
     
    //when  
    List<Client> clientList = clientRepository.findClientByEmailActiveOrCreate(email, PersonStatus.*CREATE*, PersonStatus.*ACTIVE*);  
     
    //then  
    *assertTrue*(clientList.isEmpty());  
    }  
     
    @Test  
    @Order(6)  
    void givenLogin\_itShouldGetAgentByLogin() {  
    //Given  
    String login = "login";  
     
    //When  
    Agent agent = agentRepository.findAgentByLogin(login);  
     
    //Then  
    *assertThat*(agent).isNotNull();  
    }  
     
    @Test  
    @Order(7)  
    void givenLogin\_itShouldReturnNullAgentByLogin() {  
    String login = "login569";  
     
    //When  
    Agent agent = agentRepository.findAgentByLogin(login);  
     
    //Then  
    *assertThat*(agent).isNull();  
    }  
     
    @Test  
    @Order(8)  
    void givenEmail\_itShouldGetAgentByEmail() {  
    //Given  
    String email = "emai45l@gmail.com";  
     
    //When  
    Agent agent = agentRepository.findAgentByEmail(email);  
     
    //Then  
    *assertThat*(agent).isNotNull();  
    }  
     
    @Test  
    @Order(9)  
    void givenLogin\_itShouldReturnNullAgentByEmail() {  
    String login = "login569emai45l@gmail.com";  
     
    //When  
    Agent agent = agentRepository.findAgentByLogin(login);  
     
    //Then  
    *assertThat*(agent).isNull();  
    }  
     
    @Test  
    @Order(10)  
    void givenLoginAndPassword\_itShouldGetOneAgent() {  
    //Given  
    String login = "login";  
    String password = "password";  
     
    //When  
    Agent agent = agentRepository.connectAgentByLoginAndPassword(login, password);  
     
    //Then  
    *assertThat*(agent).isNotNull();  
    }  
     
    @Test  
    @Order(11)  
    void givenLoginAndPassword\_itShouldReturnNullAgent() {  
    //Given  
    String login = "login";  
    String password = "passwordqdqs";  
     
    //When  
    Agent agent = agentRepository.connectAgentByLoginAndPassword(login, password);  
     
    //Then  
    *assertThat*(agent).isNull();  
    }  
     
    @Test  
    @Order(12)  
    void givenEmailAndPassword\_itShouldGetOneAgent() {  
    //Given  
    String email = "email@gmail.com";  
    String password = "password";  
     
    //When  
    Agent agent = agentRepository.connectAgentByEmailAndPassword(email, password);  
     
    //Then  
    *assertThat*(agent).isNotNull();  
    }  
     
    @Test  
    @Order(13)  
    void givenEmailAndPassword\_itShouldReturnNullAgent() {  
    //Given  
    String email = "email@gmail.com";  
    String password = "passwordqdqs";  
     
    //When  
    Agent agent = agentRepository.connectAgentByLoginAndPassword(email, password);  
     
    //Then  
    *assertThat*(agent).isNull();  
    }  
     
     
    @Test  
    @Order(14)  
    void givenLastNameAndFirstName\_itShouldGetAnAgent() {  
    //Given  
    String name= "agentLastName";  
    String surName = "agentFirstName";  
     
    //When  
    Agent agent = agentRepository.findAgentByLastNameAndFirstName(name, surName);  
     
    //Then  
    *assertThat*(agent).isNotNull();  
    }  
     
    @Test  
    @Order(15)  
    void givenLastNameAndFirstName\_itShouldReturnNull() {  
    //Given  
    String name= "agentLastNamqse";  
    String surName = "agentFirstName";  
     
    //When  
    Agent agent = agentRepository.findAgentByLastNameAndFirstName(name, surName);  
     
    //Then  
    *assertThat*(agent).isNull();  
    }  
     
   }
4. Tests Under services
5. package com.fabrication.agent.services;  
     
   import com.fabrication.agent.repositories.ReferenceDocumentRepository;  
   import com.fabrication.client.repositories.PersonRepository;  
   import com.fabrication.entities.Client;  
   import com.fabrication.entities.Cni;  
   import com.fabrication.entities.Passport;  
   import com.fabrication.entities.Referencedocument;  
   import com.fabrication.exceptions.ResourceNotFoundException;  
   import com.fabrication.utils.Gender;  
   import com.fabrication.utils.PersonStatus;  
   import com.fabrication.utils.StatusInTreatment;  
   import com.fabrication.utils.StatusTreatmentSystemeList;  
   import org.junit.jupiter.api.BeforeEach;  
   import org.junit.jupiter.api.Test;  
   import org.mockito.exceptions.base.MockitoException;  
   import org.springframework.data.domain.Page;  
   import org.springframework.data.domain.PageImpl;  
   import org.springframework.data.domain.PageRequest;  
   import org.springframework.data.domain.Pageable;  
     
   import java.time.Instant;  
   import java.util.\*;  
     
   import static net.bytebuddy.matcher.ElementMatchers.*is*;  
   import static org.assertj.core.api.Assertions.*assertThat*;  
   import static org.hamcrest.Matchers.\*;  
   import static org.junit.jupiter.api.Assertions.\*;  
   import static org.mockito.ArgumentMatchers.*any*;  
   import static org.mockito.Mockito.\*;  
   //@ExtendWith(MockitoExtension.class)  
   public class ReferenceDocumentCniServiceTest {  
     
     
     
     
    private ReferenceDocumentRepository<Cni> cniRepository;  
    private ReferenceDocumentRepository referenceDocumentRepository;  
    private ReferentDocumentService referentDocumentService;  
    private PersonRepository personService;  
     
    private Cni getCni() {  
    Cni cni = new Cni(  
    1L,  
    null,  
    "123456",  
    "lastName",  
    "sdvsdv",  
    Date.from(Instant.now()),  
    Gender.MALE,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.from(Instant.now()),  
    Date.from(Instant.now()),  
    "address",  
    null,  
    new Client(  
    1L,  
    "jlkfsdf@gm.de",  
    PersonStatus.ACTIVE,  
    "123456",  
    Date.from(Instant.now())  
    ),  
    null,  
    StatusTreatmentSystemeList.FORM,  
    StatusInTreatment.Waiting,  
    null  
    );  
    return cni;  
    }  
     
    private Cni getCni1() {  
    Cni cni = new Cni(  
    1L,  
    null,  
    "123456",  
    "lastName",  
    "sdvsdv",  
    Date.from(Instant.now()),  
    Gender.MALE,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.from(Instant.now()),  
    Date.from(Instant.now()),  
    "address",  
    null,  
    null,  
    null,  
    null,  
    StatusInTreatment.Waiting,  
    null  
    );  
    return cni;  
    }  
     
     
    @BeforeEach  
    void setUp() {  
    cniRepository = mock(ReferenceDocumentRepository.class);  
    referenceDocumentRepository = mock(ReferenceDocumentRepository.class);  
    personService = mock(PersonRepository.class);  
    referentDocumentService = new ReferentDocumentServiceImpl(cniRepository,null,referenceDocumentRepository, personService) ;  
    }  
     
     
    @Test  
    void itShouldSaveACni(){  
    Optional<Object> optional = Optional.of(this.getCni());  
    when(referenceDocumentRepository  
    .getCurrentReferenceDocumentByClientId(  
    this.getCni().getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    )).thenReturn(optional);  
    when(referenceDocumentRepository.save(this.getCni())).thenReturn(this.getCni1());  
    referentDocumentService.saveDocCni(this.getCni());  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenCniIdDocumentIsNull(){  
    Cni cni = getCni();  
    cni.setIdDocumentReference(null);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocCni(cni)  
    );  
    verify(referenceDocumentRepository, times(0))  
    .getCurrentReferenceDocumentByClientId(  
    cni.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(cni);  
    assertThat(exception.getMessage()).isEqualTo("Document Number is Null");  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenCniClientIsNull(){  
    Cni cni = getCni();  
    cni.setClient(null);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocCni(cni)  
    );  
    verify(referenceDocumentRepository, times(0))  
    .getCurrentReferenceDocumentByClientId(  
    cni.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(cni);  
    assertThat(exception.getMessage()).isEqualTo("Client is Null");  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenFindCniByIdClientReturnOtherType(){  
     
    Optional<Object> optional = Optional.of(new Passport());  
    Cni cni = getCni();  
    when(referenceDocumentRepository  
    .getCurrentReferenceDocumentByClientId(  
    cni.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    )).thenReturn(optional);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocCni(cni)  
    );  
    verify(referenceDocumentRepository, times(1))  
    .getCurrentReferenceDocumentByClientId(  
    cni.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(cni);  
    assertThat(exception.getMessage()).isEqualTo("Error data type is not valid");  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenFindCniByIdClientReturnNull(){  
    Cni cni = getCni();  
    Optional<Object> objectOptional = Optional.empty();  
    when(referenceDocumentRepository  
    .getCurrentReferenceDocumentByClientId(  
    cni.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    )).thenReturn(objectOptional);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocCni(cni)  
    );  
    verify(referenceDocumentRepository, times(1))  
    .getCurrentReferenceDocumentByClientId(  
    cni.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(cni);  
    assertThat(exception.getMessage()).isEqualTo("Client don't have any document");  
    }  
     
   /\* @Test  
    void itShouldUpdateACniById() {  
    Cni cni = getCni();  
    Optional<Cni> ofResult = Optional.of(cni);  
    when(cniRepository.save(any(Cni.class))).thenThrow(new ResourceNotFoundException("An error occurred"));  
    when(cniRepository.findById(any(Long.class))).thenReturn(ofResult);  
    Cni cni1 = getCni1();  
    assertThrows(ResourceNotFoundException.class, () -> referentDocumentService.updateCni(cni1, 1L));  
    verify(cniRepository).save(any());  
    verify(cniRepository).findById(any(Long.class));  
    }\*/  
     
    @Test  
    void itShouldThrowAnExceptionForListOfReferenceDocumentEmpty() {  
     
    //Given  
    List<Object> objectList = new ArrayList<>();  
     
    //When  
    *when*(referenceDocumentRepository.getReferenceDocumentByClientId(*any*(Long.class))).thenReturn(objectList);  
    Throwable exception = *assertThrows*(  
    ResourceNotFoundException.class,  
    () -> referentDocumentService.getReferenceDocumentByClientId(*any*(Long.class))  
    );  
     
    //Then  
    *assertThat*(exception.getMessage()).isEqualTo("Element not exist");  
      
      
    }  
     
    @Test  
    void itShouldReturnAListOfReferenceDocumentForClient() {  
     
    //When  
    List<Object> objectList = new ArrayList<>();  
    objectList.add(this.getCni());  
    objectList.add(this.getCni1());  
    when(referenceDocumentRepository.getReferenceDocumentByClientId(any(Long.class)))  
    .thenReturn(objectList);  
    List<Object> objectListInService = referentDocumentService.getReferenceDocumentByClientId(any(Long.class));  
     
    //Then  
    assertThat(objectListInService.size()).isEqualTo(objectList.size());  
    }  
     
    @Test  
    void itShouldThrowAnExceptionForCurrentReferenceDocument() {  
    Optional<Object> objectOptional = Optional.empty();  
    when(referenceDocumentRepository.getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)).thenReturn(objectOptional);  
     
    //When  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    () -> referentDocumentService.getCurrentReferenceDocumentByClientId(1L)  
    );  
     
    //Then  
    assertThat(exception.getMessage()).isEqualTo("Element not exist");  
    }  
     
    @Test  
    void itShouldReturnCurrentReferenceDocument() {  
     
    //Given  
    Optional<Object> optional = Optional.of(this.getCni());  
    //When  
    when(referenceDocumentRepository.getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH))  
    .thenReturn(optional);  
    Cni cni = (Cni) referentDocumentService.getCurrentReferenceDocumentByClientId(1L);  
     
    //Then  
    assertThat(cni).isNotNull();  
    }  
     
    @Test  
    void itShouldGenerateReferenceFormId(){  
    String referenceFormId = referentDocumentService.generateReferenceFormId(1L);  
    assertThat(referenceFormId.length()).isEqualTo(7);  
    assertTrue(referenceFormId.endsWith("1"));  
    }  
     
    @Test  
    void itShouldInitCni(){  
    Client client = new Client(  
    1L,  
    "sf@gmail.co",  
    PersonStatus.ACTIVE,  
    "123456",  
    Date.from(Instant.now())  
    );  
    when(personService.findClientById(1L)).thenReturn(client);  
    when(referenceDocumentRepository.save(any())).thenReturn(any());  
    referentDocumentService.initCni(1L);  
    verify(personService, times(1)).findClientById(1L);  
    verify(referenceDocumentRepository, times(1)).save(any());  
    }  
     
    @Test  
    void itShouldThrowAnExceptionWhenInitCni(){  
    when(personService.findClientById(1L)).thenThrow(new ResourceNotFoundException("Client don't exist"));  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.initCni(1L)  
    );  
    assertThat(exception.getMessage()).isEqualTo("Client don't exist");  
    verify(referenceDocumentRepository, times(0)).save(any());  
    }  
     
    @Test  
    void itShouldFindAllReferenceByStatusTreatmentSystemListAndPage() {  
    //Given  
    List<Referencedocument> referenceDocumentList = new ArrayList<>();  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    Page<Referencedocument> referenceDocumentPageMock = new PageImpl<>(  
    referenceDocumentList, PageRequest.of(0,5),5  
    );  
     
    //When  
    when(  
    referenceDocumentRepository.findAll(StatusTreatmentSystemeList.BUILD,PageRequest.of(0,5))  
    )  
    .thenReturn(referenceDocumentPageMock);  
    Page<Referencedocument>referencedocumentPage = referentDocumentService  
    .findAllReferenceByStatusTreatmentSystemListAndPage(StatusTreatmentSystemeList.BUILD,0,5);  
     
    //Then  
    assertThat(referencedocumentPage.isEmpty()).isFalse();  
    assertThat(referencedocumentPage.getSize()).isEqualTo(5);  
    assertThat(referencedocumentPage.getNumberOfElements()).isEqualTo(10);  
    }  
     
    @Test  
    void itShouldThrowAnExceptionFindAllReferenceByStatusTreatmentSystemListAndPage() {  
     
    //When  
    when(  
    referenceDocumentRepository.findAll(StatusTreatmentSystemeList.BUILD,PageRequest.of(0,5))  
    )  
    .thenThrow(MockitoException.class);  
     
    ResourceNotFoundException exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService  
    .findAllReferenceByStatusTreatmentSystemListAndPage(  
    StatusTreatmentSystemeList.BUILD,  
    0,  
    5  
    )  
    ) ;  
     
    //Then  
    assertThat(exception.getMessage()).isEqualTo("Error While getting data");  
    }  
     
    @Test  
    void itShouldFindAllReferenceInBuildingStepByStatusInTreatment() {  
    //Given  
    List<Referencedocument> referenceDocumentList = new ArrayList<>();  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    referenceDocumentList.add(this.getCni1());  
    referenceDocumentList.add(this.getCni());  
    Page<Referencedocument> referenceDocumentPageMock = new PageImpl<>(  
    referenceDocumentList, PageRequest.of(0,5),5  
    );  
     
    //When  
    when(  
    referenceDocumentRepository.findAll(  
    StatusTreatmentSystemeList.BUILD,  
    StatusInTreatment.Ready,  
    PageRequest.of(0,5))  
    )  
    .thenReturn(referenceDocumentPageMock);  
    Page<Referencedocument>referencedocumentPage = referentDocumentService  
    .findAllReferenceInBuildingStepByStatusInTreatmentAndPage(  
    StatusInTreatment.Ready,  
    0,  
    5);  
     
    //Then  
    assertThat(referencedocumentPage.isEmpty()).isFalse();  
    assertThat(referencedocumentPage.getSize()).isEqualTo(5);  
    assertThat(referencedocumentPage.getNumberOfElements()).isEqualTo(10);  
    }  
     
   }
6. package com.fabrication.agent.services;  
     
     
   import com.fabrication.agent.repositories.ReferenceDocumentRepository;  
   import com.fabrication.client.repositories.PersonRepository;  
   import com.fabrication.entities.Client;  
   import com.fabrication.entities.Cni;  
   import com.fabrication.entities.Passport;  
   import com.fabrication.exceptions.ResourceNotFoundException;  
   import com.fabrication.utils.Gender;  
   import com.fabrication.utils.PersonStatus;  
   import com.fabrication.utils.StatusInTreatment;  
   import com.fabrication.utils.StatusTreatmentSystemeList;  
   import org.assertj.core.api.Assertions;  
   import org.junit.jupiter.api.BeforeEach;  
   import org.junit.jupiter.api.Test;  
   import org.junit.jupiter.api.extension.ExtendWith;  
   import org.mockito.junit.jupiter.MockitoExtension;  
     
   import java.time.Instant;  
   import java.util.Date;  
   import java.util.Optional;  
     
   import static org.assertj.core.api.Java6Assertions.*assertThat*;  
   import static org.junit.jupiter.api.Assertions.*assertThrows*;  
   import static org.mockito.ArgumentMatchers.*any*;  
   import static org.mockito.Mockito.\*;  
   //@ExtendWith(MockitoExtension.class)  
   public class ReferenceDocumentPassportServiceTest {  
     
    private ReferenceDocumentRepository<Passport> passportRepository;  
    private ReferentDocumentService referentDocumentService;  
     
    private ReferenceDocumentRepository referenceDocumentRepository;  
    private PersonRepository personService;  
     
     
    private Passport getPassport() {  
    return new Passport(  
    2L,  
    null,  
    "123456",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.*from*(Instant.*now*()),  
    Date.*from*(Instant.*now*()),  
    "address",  
    null,  
    new Client(  
    2L,  
    "kljsdklf@qsd.qsd",  
    PersonStatus.*ACTIVE*,  
    "123456",  
    Date.*from*(Instant.*now*())  
    ),  
    null,  
    StatusTreatmentSystemeList.*FORM*,  
    StatusInTreatment.*Waiting*,  
    null,  
    "Country",  
    null  
    );  
    }  
     
    private Passport getPassport1() {  
    return new Passport(  
    2L,  
    null,  
    "123456",  
    "lastName",  
    "sdvsdv",  
    Date.*from*(Instant.*now*()),  
    Gender.*MALE*,  
    "profession",  
    "nameOfFather",  
    "nameOfMother",  
    Date.from(Instant.now()),  
    Date.from(Instant.now()),  
    "address",  
    null,  
    new Client(  
    2L,  
    "kljsdklf@qsd.qsd",  
    PersonStatus.ACTIVE,  
    "123456",  
    Date.from(Instant.now())  
    ),  
    null,  
    StatusTreatmentSystemeList.FORM,  
    StatusInTreatment.Waiting,  
    null,  
    "Country",  
    null  
    );  
    }  
     
     
     
    @BeforeEach  
    void setUp() {  
    passportRepository = mock(ReferenceDocumentRepository.class);  
    personService = mock(PersonRepository.class);  
    referenceDocumentRepository = mock(ReferenceDocumentRepository.class);  
    referentDocumentService = new ReferentDocumentServiceImpl(null,passportRepository,referenceDocumentRepository, personService) ;  
    }  
     
    @Test  
    void itShouldSaveAPassPort(){  
    when(referenceDocumentRepository  
    .getCurrentReferenceDocumentByClientId(  
    this.getPassport().getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    )).thenReturn(Optional.of(this.getPassport()));  
    when(referenceDocumentRepository.save(this.getPassport())).thenReturn(this.getPassport1());  
    referentDocumentService.saveDocPassport(this.getPassport());  
    // verify(referenceDocumentRepository, times(1)).save(this.getPassport());  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenPassPortIdDocumentIsNull(){  
    Passport passport = getPassport();  
    passport.setIdDocumentReference(null);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocPassport(passport)  
    );  
    verify(referenceDocumentRepository, times(0))  
    .getCurrentReferenceDocumentByClientId(  
    passport.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(passport);  
    Assertions.assertThat(exception.getMessage()).isEqualTo("Document Number is Null");  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenCniClientIsNull(){  
    Passport passport = getPassport();  
    passport.setClient(null);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocPassport(passport)  
    );  
    verify(referenceDocumentRepository, times(0))  
    .getCurrentReferenceDocumentByClientId(  
    passport.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(passport);  
    Assertions.assertThat(exception.getMessage()).isEqualTo("Client is Null");  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenFindPassPortByIdClientReturnOtherType(){  
    Passport passport = getPassport();  
    when(referenceDocumentRepository  
    .getCurrentReferenceDocumentByClientId(  
    passport.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    )).thenReturn(Optional.of(new Cni()));  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocPassport(passport)  
    );  
    verify(referenceDocumentRepository, times(1))  
    .getCurrentReferenceDocumentByClientId(  
    passport.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(passport);  
    Assertions.assertThat(exception.getMessage()).isEqualTo("Error data type is not valid");  
    }  
     
    @Test  
    public void itShouldThrowExceptionWhenFindPassPortByIdClientReturnNull(){  
    Passport passport = this.getPassport();  
    Optional<Object> objectOptional = null;  
    when(referenceDocumentRepository  
    .getCurrentReferenceDocumentByClientId(  
    1L,  
    StatusTreatmentSystemeList.FINISH  
    )).thenReturn(objectOptional);  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.saveDocPassport(passport)  
    );  
    verify(referenceDocumentRepository, times(1))  
    .getCurrentReferenceDocumentByClientId(  
    passport.getIdDocumentReference(),  
    StatusTreatmentSystemeList.FINISH  
    );  
    verify(referenceDocumentRepository, times(0))  
    .save(passport);  
    Assertions.assertThat(exception.getMessage()).isEqualTo("Client don't have any document");  
    }  
     
    @Test  
    void itShouldUpdateAPassportById() {  
    Passport passport = getPassport();  
     
    Optional<Passport> ofResult = Optional.of(passport);  
    when(passportRepository.save(any())).thenThrow(new ResourceNotFoundException("An error occurred"));  
    when(passportRepository.findById(any())).thenReturn(ofResult);  
     
    Passport passport1 = getPassport1();  
    assertThrows(ResourceNotFoundException.class, () -> referentDocumentService.updatePassport(passport1, 1L));  
    verify(passportRepository).save(any());  
    verify(passportRepository).findById(any());  
    }  
     
    @Test  
    void itShouldInitPassPort(){  
    Client client = new Client(  
    1L,  
    "sf@gmail.co",  
    PersonStatus.ACTIVE,  
    "123456",  
    Date.from(Instant.now())  
    );  
    when(personService.findClientById(1L)).thenReturn(client);  
    when(referenceDocumentRepository.save(any())).thenReturn(any());  
    referentDocumentService.initPassPort(1L);  
    verify(personService, times(1)).findClientById(1L);  
    verify(referenceDocumentRepository, times(1)).save(any());  
    }  
     
    @Test  
    void itShouldThrowAnExceptionWhenInitPassPort(){  
    when(personService.findClientById(1L)).thenThrow(new ResourceNotFoundException("Client don't exist"));  
    Throwable exception = assertThrows(  
    ResourceNotFoundException.class,  
    ()->referentDocumentService.initPassPort(1L)  
    );  
    Assertions.assertThat(exception.getMessage()).isEqualTo("Client don't exist");  
    verify(referenceDocumentRepository, times(0)).save(any());  
    }  
     
     
     
     
     
     
     
     
     
   }
7. package com.fabrication.client.services;  
     
   import com.fabrication.client.repositories.PersonRepository;  
   import com.fabrication.entities.Client;  
   import com.fabrication.services.EmailService;  
   import com.fabrication.services.EmailServiceImpl;  
   import com.fabrication.utils.PersonStatus;  
   import org.junit.jupiter.api.BeforeEach;  
   import org.junit.jupiter.api.DisplayName;  
   import org.junit.jupiter.api.Test;  
     
   import java.sql.Date;  
   import java.time.Instant;  
   import java.util.ArrayList;  
   import java.util.List;  
     
   import static org.assertj.core.api.Assertions.*assertThat*;  
   import static org.junit.jupiter.api.Assertions.\*;  
   import static org.mockito.ArgumentMatchers.*any*;  
   import static org.mockito.Mockito.\*;  
     
   public class PersonServiceClientTest {  
    private PersonRepository clientRepositoryMock;  
     
    private EmailService emailServiceMock;  
    private PersonService personServiceTest;  
     
    @BeforeEach  
    void setUp() {  
    clientRepositoryMock = *mock*(PersonRepository.class);  
    emailServiceMock = *mock*(EmailServiceImpl.class);  
    personServiceTest = new PersonServiceImpl(clientRepositoryMock, emailServiceMock) ;  
    }  
     
    @Test  
    @DisplayName("it should save a client")  
    void itShouldSaveAClient() {  
    Client client = new Client(  
    1L,  
    "to@gmail.com",  
    PersonStatus.*CREATE*,  
    "toto",  
    Date.*from*(Instant.*now*()));  
    *when*(clientRepositoryMock.save(*any*(Client.class))).thenReturn(client);  
    personServiceTest.savePersonClient(client);  
    *verify*(emailServiceMock,*times*(1)).generateValidationCode();  
    *verify*(clientRepositoryMock,*times*(1)).save(client);  
    // verify(emailServiceMock,times(1)).sendSimpleMessage(client.getEmail(), "Validation",client.getValidationCode());  
    }  
     
    @Test  
    @DisplayName("it should throw and exception when email is null")  
    void itShouldThrowExceptionWhenEmailIsNull() {  
    Throwable exception = *assertThrows*(  
    Exception.class,  
    ()-> personServiceTest.connectClient(null)  
    );  
    *assertThat*(exception.getMessage()).isEqualTo("Email is invalid");  
    }  
     
    @Test  
    @DisplayName("it should connect a client to continuous or verified identification")  
    void itShouldConnectClientToContinuousOrVerifiedIdentification() {  
    List<Client> clientList = new ArrayList<>();  
    Client client = new Client(  
    1L,  
    "address@email.com",  
    PersonStatus.*ACTIVE*,  
    "toto",  
    Date.*from*(Instant.*now*()));  
    clientList.add(client);  
    *when*(  
    clientRepositoryMock  
    .findClientByEmailActiveOrCreate("address@email.com", PersonStatus.*CREATE*, PersonStatus.*ACTIVE*))  
    .thenReturn(clientList);  
    personServiceTest.connectClient("address@email.com");  
    *assertThat*(clientList.get(0).getEmail()).isEqualTo("address@email.com");  
    }  
     
    @Test  
    @DisplayName("it should create a client")  
    void itShouldCreateAClient() {  
    List<Client> clientList = new ArrayList<>();  
    Client client;  
    *when*(  
    clientRepositoryMock  
    .findClientByEmailActiveOrCreate("address@email.com", PersonStatus.*CREATE*, PersonStatus.*ACTIVE*))  
    .thenReturn(clientList);  
    client = personServiceTest.connectClient("address@email.com");  
    *verify*(emailServiceMock,*times*(1)).generateValidationCode();  
    *assertThat*(client.getEmail()).isEqualTo(client.getEmail());  
    }  
     
    @Test  
    @DisplayName("it should disable a client")  
    void itShouldDisableAClient() {  
    Client client = new Client(  
    1L,  
    "address@email.com",  
    PersonStatus.*INACTIVE*,  
    "toto",  
    Date.*from*(Instant.*now*()));  
    *when*(clientRepositoryMock.findClientByEmail("address@email.com", PersonStatus.*ACTIVE*)).thenReturn(client);  
    *when*(  
    clientRepositoryMock  
    .save(client))  
    .thenReturn(client);  
    personServiceTest.disableClient("address@email.com");  
    *verify*(clientRepositoryMock,*times*(1))  
    .findClientByEmail(client.getEmail(), PersonStatus.*ACTIVE*);  
    *verify*(clientRepositoryMock,*times*(1))  
    .save(client);  
    *assertThat*(client.getPersonStatus()).isEqualTo(PersonStatus.*INACTIVE*);  
    }  
     
    @Test  
    @DisplayName("It should verified validation code")  
    void itShouldValidateClientCode() {  
    String code ="123456";  
    String email = "address@email.com";  
    Client client = new Client(  
    1L,  
    "address@email.com",  
    PersonStatus.*ACTIVE*,  
    "toto",  
    Date.*from*(Instant.*now*())  
    );  
    *when*(clientRepositoryMock.findClientByEmailAndValidationCode(email, code, PersonStatus.*CREATE*)).thenReturn(client);  
    boolean verificationValue = personServiceTest.codeClientValidation(email, code);  
    *verify*(clientRepositoryMock,*times*(1)).save(client);  
    *assertTrue*(verificationValue);  
    }  
     
    @Test  
    @DisplayName("It should't verified validation code")  
    void itShouldNotValidateClientCode() {  
    String code ="123456";  
    String email = "address@email.com";  
    *when*(clientRepositoryMock.findClientByEmailAndValidationCode(email, code, PersonStatus.*CREATE*)).thenReturn(null);  
    boolean verificationValue = personServiceTest.codeClientValidation(email, code);  
    *assertFalse*(verificationValue);  
    }  
     
   }
8. package com.fabrication.client.services;  
     
   import com.fabrication.entities.Agent;  
   import com.fabrication.client.repositories.PersonRepository;  
   import com.fabrication.utils.LoginBean;  
   import com.fabrication.utils.PersonStatus;  
   import org.junit.jupiter.api.BeforeEach;  
   import org.junit.jupiter.api.DisplayName;  
   import org.junit.jupiter.api.Order;  
   import org.junit.jupiter.api.Test;  
   import org.mockito.exceptions.base.MockitoException;  
     
   import static org.assertj.core.api.Assertions.*assertThat*;  
   import static org.junit.jupiter.api.Assertions.assertEquals;  
   import static org.junit.jupiter.api.Assertions.assertThrows;  
   import static org.mockito.ArgumentMatchers.any;  
   import static org.mockito.Mockito.\*;  
     
   public class PersonServiceAgentTest {  
     
    private PersonRepository agentRepositoryMock;  
    private PersonService personServiceTest;  
     
    @BeforeEach  
    void setUp() {  
    agentRepositoryMock = *mock*(PersonRepository.class);  
    personServiceTest = new PersonServiceImpl(agentRepositoryMock,null) ;  
    }  
     
    @Test  
    @DisplayName("it should save and agent")  
    void itShouldSaveAndAgent() {  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.*ACTIVE*,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.save(any(Agent.class))).thenReturn(agent);  
    personServiceTest.savePersonAgent(agent);  
    verify(agentRepositoryMock,times(1)).save(agent);  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when login is null")  
    void itShouldThrowExceptionWhenLoginNull() {  
    Agent agent = new Agent(  
    null,  
    "String email",  
    PersonStatus.ACTIVE,  
    "agentLastName",  
    "agentFirstName",  
    null,  
    "password");  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("Login is null", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when email is null")  
    void itShouldThrowExceptionWhenEmailIsNull() {  
    Agent agent = new Agent(  
    null,  
    null,  
    PersonStatus.ACTIVE,  
    "agentLastName",  
    "agentFirstName",  
    "Login",  
    "password");  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("Email is null", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when password is null")  
    void itShouldThrowExceptionWhenPasswordIsNull() {  
    Agent agent = new Agent(  
    null,  
    "null",  
    PersonStatus.ACTIVE,  
    "agentLastName",  
    "agentFirstName",  
    "Login",  
    null);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("Password is null", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when LastName is null")  
    void itShouldThrowExceptionWhenLastNameIsNull() {  
    Agent agent = new Agent(  
    null,  
    "xvdfvdfsv@gmail.com",  
    PersonStatus.ACTIVE,  
    null,  
    "agentFirstName",  
    "Login",  
    "password");  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("LastName is null", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when firstname is null")  
    void itShouldThrowExceptionWhenFirstNameIsNull() {  
    Agent agent = new Agent(  
    null,  
    "xvdfvdfsv@gmail.com",  
    PersonStatus.ACTIVE,  
    "agentLastName",  
    null,  
    "Login",  
    "password");  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("FirstName is null", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when login already exist")  
    void itShouldThrowExceptionWhenLoginAlreadyExist() {  
    Agent agent = new Agent(  
    null,  
    "to@gmail.com",  
    PersonStatus.ACTIVE,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.findAgentByLogin(any(String.class))).thenReturn(agent);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("Login "+agent.getLogin()+" already use", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when email already exist")  
    @Order(1)  
    void itShouldThrowExceptionWhenEmailAlreadyExist() {  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.ACTIVE,  
    "toto",  
    "toto",  
    "loginh",  
    "password");  
    when(agentRepositoryMock.findAgentByEmail(any(String.class))).thenReturn(agent);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals("Email "+agent.getEmail()+" already use", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when lastname and firstname already exist")  
    @Order(1)  
    void itShouldThrowExceptionWhenFirstNameAndLastNamrAlreadyExist() {  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.ACTIVE,  
    "toto",  
    "toto",  
    "loginh",  
    "password");  
    when(agentRepositoryMock.findAgentByLastNameAndFirstName(any(String.class), any(String.class)))  
    .thenReturn(agent);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.savePersonAgent(agent)  
    );  
    assertEquals(  
    "Agent with LastName " +agent.getAgentLastName() +" and FirstName "+agent.getAgentFirstName() +"already exist",  
    exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("It should connect an agent with login and password")  
    void itShouldConnectAnAgentWithLoginAndPassword() {  
    LoginBean loginBean = new LoginBean("login","password");  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.ACTIVE,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(agent);  
    assertThat(personServiceTest.connectAgent(loginBean)).isNotNull();  
    }  
     
    @Test  
    @DisplayName("It should connect an agent with email and password")  
    void itShouldConnectAnAgentWithEmailAnPassword() {  
    LoginBean loginBean = new LoginBean("login@gmail.com","password");  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.ACTIVE,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(null);  
    when(agentRepositoryMock.connectAgentByEmailAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(agent);  
    assertThat(personServiceTest.connectAgent(loginBean)).isNotNull();  
    }  
     
    @Test  
    @DisplayName("It should throw an exception when connect an agent because login is empty")  
    void itShouldThrowAnExceptionWhenConnectAnAgentBecauseLoginIsEmpty() {  
    LoginBean loginBean = new LoginBean("","password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenThrow(new MockitoException("Login is null"));  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.connectAgent(loginBean)  
    );  
    assertEquals("Login is null", exception.getMessage());  
    }  
     
     
    @Test  
    @DisplayName("It should throw an exception when connect an agent because password is null")  
    void itShouldThrowAnExceptionWhenConnectAnAgentBecausePasswordIsNull() {  
    LoginBean loginBean = new LoginBean("login",null);  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.ACTIVE,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(agent);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.connectAgent(loginBean)  
    );  
    assertEquals("Password is null", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("It should throw an exception when connect an agent because agent don't exist")  
    void itShouldThrowAnExceptionWhenConnectAnAgentBecauseAgentDoNotExist() {  
    LoginBean loginBean = new LoginBean("login","password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(null);  
    when(agentRepositoryMock.connectAgentByEmailAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(null);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.connectAgent(loginBean)  
    );  
    assertEquals("Login and/or Password invalid", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("It should throw an exception when connect an agent because agent account is locked")  
    void itShouldThrowAnExceptionWhenConnectAnAgentBecauseAgentAccountIsLocked() {  
    LoginBean loginBean = new LoginBean("login","password");  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.INACTIVE,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(agent);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.connectAgent(loginBean)  
    );  
    assertEquals("This account is locked", exception.getMessage());  
    }  
     
    @Test  
    @DisplayName("It should throw an exception when connect an agent because agent account is locked (Email)")  
    void itShouldThrowAnExceptionWhenConnectAnAgentBecauseAgentAccountIsLocked2() {  
    LoginBean loginBean = new LoginBean("login","password");  
    Agent agent = new Agent(  
    1L,  
    "to@gmail.com",  
    PersonStatus.INACTIVE,  
    "toto",  
    "toto",  
    "login",  
    "password");  
    when(agentRepositoryMock.connectAgentByLoginAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(null);  
    when(agentRepositoryMock.connectAgentByEmailAndPassword(loginBean.getLogin(), loginBean.getPassword()))  
    .thenReturn(agent);  
    Throwable exception = assertThrows(  
    Exception.class,  
    () -> personServiceTest.connectAgent(loginBean)  
    );  
    assertEquals("This account is locked", exception.getMessage());  
    }  
   }
9. package com.fabrication.client.services;  
     
   import com.fabrication.services.EmailService;  
   import com.fabrication.services.EmailServiceImpl;  
   import org.junit.jupiter.api.BeforeEach;  
   import org.junit.jupiter.api.DisplayName;  
   import org.junit.jupiter.api.Test;  
   import org.springframework.beans.factory.annotation.Value;  
   import org.springframework.mail.SimpleMailMessage;  
   import org.springframework.mail.javamail.JavaMailSender;  
   import org.springframework.mail.javamail.JavaMailSenderImpl;  
     
   import static org.assertj.core.api.Assertions.*assertThat*;  
   import static org.junit.jupiter.api.Assertions.*assertThrows*;  
   import static org.junit.jupiter.api.Assertions.*assertTrue*;  
   import static org.mockito.Mockito.\*;  
     
   class EmailServiceImplTest {  
     
    private EmailService emailService;  
     
    @Value("${spring.mail.username}")  
    private String sender;  
     
    private JavaMailSender javaMailSenderMock;  
     
    @BeforeEach  
    void setUp(){  
    javaMailSenderMock = mock(JavaMailSenderImpl.class);  
    emailService = new EmailServiceImpl(javaMailSenderMock);  
    }  
     
    @Test  
    @DisplayName("it should send simple message")  
    void itShouldSendSimpleMessageTest() {  
    // System.err.println(sender);  
    String to = "addres@gmail.com";  
    String subject = "Validation";  
    String text = "123456";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p>For security reason, you're required to use the following "  
    + "One Time Password to login:</p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    emailService.sendSimpleMessage(to,subject,text);  
    verify(javaMailSenderMock,times(1)).send(message);  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when receiver is invalid")  
    void itShouldThrowAnExceptionWhenReceiverIsInvalid() {  
    String to = "hghg";  
    String subject = "Validation";  
    String text = "123456";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p>For security reason, you're required to use the following "  
    + "One Time Password to login:</p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    Throwable exception = assertThrows(  
    Exception.class,  
    ()-> emailService.sendSimpleMessage(to, subject, text)  
    );  
    assertThat(exception.getMessage())  
    .isEqualTo("Receiver address is not valid");  
    verify(javaMailSenderMock,times(0)).send(message);  
    }  
     
    @Test  
    @DisplayName("it should throw an exception when subject is empty")  
    void itShouldThrowAnExceptionWhenSubjectIsEmpty() {  
    String to = "address@gmail.com";  
    String subject = "";  
    String text = "123456";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p>For security reason, you're required to use the following "  
    + "One Time Password to login:</p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    Throwable exception = assertThrows(  
    Exception.class,  
    ()-> emailService.sendSimpleMessage(to, subject, text)  
    );  
    assertThat(exception.getMessage())  
    .isEqualTo("Subject is not valid");  
    verify(javaMailSenderMock,times(0)).send(message);  
    }  
     
     
    @Test  
    @DisplayName("it should throw an exception when code is not valid")  
    void itShouldThrowAnExceptionWhenCodeIsNotValid() {  
    String to = "address@gmail.com";  
    String subject = "Validation";  
    String text = "12345jkgjkhgkj";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p>For security reason, you're required to use the following "  
    + "One Time Password to login:</p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    Throwable exception = assertThrows(  
    Exception.class,  
    ()-> emailService.sendSimpleMessage(to, subject, text)  
    );  
    assertThat(exception.getMessage())  
    .isEqualTo("Validation Code is not valid");  
    verify(javaMailSenderMock,times(0)).send(message);  
    }  
     
     
     
    @Test  
    @DisplayName("it should send mail to notified client")  
    void itShouldSendEmailToNotifiedClientTest() {  
    // System.err.println(sender);  
    String to = "addres@gmail.com";  
    String subject = "Emition";  
    String text = "125478";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    emailService.sendMailToEmition(to,subject,text);  
    verify(javaMailSenderMock,times(1)).send(message);  
    }  
     
    @Test  
    void itShouldThrowAnExceptionWhenReceiverIsInvalidSendMailToEmit() {  
    String to = "hghg";  
    String subject = "Validation";  
    String text = "123456";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    Throwable exception = assertThrows(  
    Exception.class,  
    ()-> emailService.sendMailToEmition(to, subject, text)  
    );  
    assertThat(exception.getMessage())  
    .isEqualTo("Receiver address is not valid");  
    verify(javaMailSenderMock,times(0)).send(message);  
    }  
     
    @Test  
    void itShouldThrowAnExceptionWhenSubjectIsEmptySendMailToEmition() {  
    String to = "address@gmail.com";  
    String subject = "";  
    String text = "123456";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    Throwable exception = assertThrows(  
    Exception.class,  
    ()-> emailService.sendMailToEmition(to, subject, text)  
    );  
    assertThat(exception.getMessage())  
    .isEqualTo("Subject is not valid");  
    verify(javaMailSenderMock,times(0)).send(message);  
    }  
     
     
    @Test  
    void itShouldThrowAnExceptionWhenCodeIsNotValidSendMailToEmition() {  
    String to = "address@gmail.com";  
    String subject = "Validation";  
    String text = "12345jkgjkhgkj";  
    SimpleMailMessage message = new SimpleMailMessage();  
    message.setFrom(sender);  
    message.setTo(to);  
    message.setSubject(subject);  
    message.setText(  
    "<p>Hello, </p>"  
    + "<p><b>" + text + "</b></p>"  
    + "<br>"  
    );  
    Throwable exception = assertThrows(  
    Exception.class,  
    ()-> emailService.sendMailToEmition(to, subject, text)  
    );  
    assertThat(exception.getMessage())  
    .isEqualTo("Validation Code is not valid");  
    verify(javaMailSenderMock,times(0)).send(message);  
    }  
     
    @Test  
    @DisplayName("it should generate validation code")  
    void itShouldGenerateValidationCode() {  
    assertTrue(emailService.generateValidationCode().matches("^[0123456789]{6}"));  
    }  
   }
10. package com.fabrication.client.services;  
      
    import com.fabrication.agent.repositories.ReferenceDocumentRepository;  
    import com.fabrication.entities.Client;  
    import com.fabrication.entities.Cni;  
    import com.fabrication.entities.Passport;  
    import com.fabrication.exceptions.ResourceNotFoundException;  
    import com.fabrication.utils.Gender;  
    import com.fabrication.utils.PersonStatus;  
    import com.fabrication.utils.StatusInTreatment;  
    import com.fabrication.utils.StatusTreatmentSystemeList;  
    import com.itextpdf.text.DocumentException;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.springframework.mock.web.MockHttpServletResponse;  
      
    import java.io.IOException;  
    import java.time.Instant;  
    import java.util.Date;  
    import java.util.Optional;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.junit.jupiter.api.Assertions.*assertThrows*;  
    import static org.mockito.ArgumentMatchers.*any*;  
    import static org.mockito.BDDMockito.*given*;  
    import static org.mockito.Mockito.*mock*;  
    import static org.mockito.Mockito.*when*;  
      
    public class BuildClientPDFTest {  
     private BuildClientPDFService buildClientPDFService;  
     private PersonService personService;  
      
     private ReferenceDocumentRepository referencedocumentRepository;  
      
     private Cni getCni() {  
     Cni cni = new Cni();  
     cni.setIdDocumentReference(12L);  
     cni.setFirstName("updaterepoDTFB");  
     cni.setAddress("updaterepoAdd");  
     cni.setLastName("updaterepoLN");  
     cni.setDateOfBirth(Date.*from*(Instant.*now*()));  
     cni.setDeliveryDate(Date.*from*(Instant.*now*()));  
     cni.setGender(Gender.*MALE*);  
     cni.setExpirationDate(Date.*from*(Instant.*now*()));  
     cni.setDocumentNumber(null);  
     cni.setNameOfFather("updaterepoFa");  
     cni.setNameOfMother("updaterepoMo");  
     return cni;  
     }  
      
     private Passport getPassport(){  
     Passport passport= new Passport(  
     2L,  
     null,  
     null,  
     "passport1Firstname9",  
     "passportLastName9",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "INGENIEUR",  
     "passportFATHER6",  
     "passportMother6",  
     null,  
     null,  
     "Mendong",  
     null,  
     null,  
     null,  
     null,  
     null,  
     null  
     );  
     return passport;  
     }  
      
     @BeforeEach  
     void setUp(){  
     personService = *mock*(PersonServiceImpl.class);  
     referencedocumentRepository = *mock*(ReferenceDocumentRepository.class);  
     buildClientPDFService = new BuildClientPDFServiceImpl(personService, referencedocumentRepository);  
     }  
      
       
    }
11. package com.fabrication.build.services;  
      
    import com.fabrication.agent.repositories.ReferenceDocumentRepository;  
    import com.fabrication.entities.\*;  
    import com.fabrication.utils.Gender;  
    import com.fabrication.utils.PersonStatus;  
    import com.fabrication.utils.StatusInTreatment;  
    import com.fabrication.utils.StatusTreatmentSystemeList;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.mockito.exceptions.base.MockitoException;  
    import org.springframework.data.domain.Page;  
    import org.springframework.data.domain.PageImpl;  
    import org.springframework.data.domain.PageRequest;  
      
    import java.time.Instant;  
    import java.util.ArrayList;  
    import java.util.Date;  
    import java.util.List;  
    import java.util.Optional;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.junit.jupiter.api.Assertions.\*;  
    import static org.mockito.ArgumentMatchers.*any*;  
    import static org.mockito.ArgumentMatchers.*anyInt*;  
    import static org.mockito.Mockito.\*;  
      
    class BuildServiceImplTest {  
     private BuildService buildService;  
     private ReferenceDocumentRepository referenceDocumentRepository;  
      
     @BeforeEach  
     void setUp(){  
     referenceDocumentRepository = *mock*(ReferenceDocumentRepository.class);  
     buildService = new BuildServiceImpl(referenceDocumentRepository);  
     }  
      
     private Imageadditionaldocument getImageAdditionalDocument(){  
     return new Imageadditionaldocument(  
     1L,  
     "jhvjhvjh",  
     Date.*from*(Instant.*now*()),  
     new Agent(),  
     new Client(),  
     new Cni()  
     );  
     }  
      
     private Page<Referencedocument> listDocReadyToBuildData(){  
     Cni cni = getReferenceDocument();  
     Cni cni1 = new Cni(  
     89L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     Cni cni2 = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     198L,  
     "kjjlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     Cni cni3 = new Cni(  
     889L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     7L,  
     "jlkfsf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     List<Referencedocument> referencedocumentList = new ArrayList<>();  
     referencedocumentList.add(cni);  
     referencedocumentList.add(cni1);  
     referencedocumentList.add(cni2);  
     referencedocumentList.add(cni3);  
      
     return new PageImpl<Referencedocument>(referencedocumentList, PageRequest.*of*(0, 2),4L);  
      
     }  
      
     private Page<Referencedocument> listDocInRealTimeBuildData(){  
     Cni cni = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni1 = new Cni(  
     89L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni2 = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     198L,  
     "kjjlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni3 = new Cni(  
     889L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     7L,  
     "jlkfsf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     List<Referencedocument> referencedocumentList = new ArrayList<>();  
     referencedocumentList.add(cni);  
     referencedocumentList.add(cni1);  
     referencedocumentList.add(cni2);  
     referencedocumentList.add(cni3);  
      
     return new PageImpl<Referencedocument>(referencedocumentList, PageRequest.*of*(0, 2),4L);  
      
     }  
      
     @Test  
     void itShouldReturnPageListOfDocReadyToBuild() {  
     *when*(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     PageRequest.*of*(0,2)  
     )  
     ).thenReturn(this.listDocReadyToBuildData());  
     Page<Referencedocument> data = buildService.listDocReadyToBuild(0,2);  
     *verify*(referenceDocumentRepository, *times*(1)).findAll(  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     PageRequest.*of*(0,2)  
     );  
     *assertThat*(data.isEmpty()).isFalse();  
     *assertThat*(data.getContent()).isNotEmpty();  
     }  
      
     @Test  
     void itShouldReturnAnEmptyPageListOfDocReadyToBuild() {  
     *when*(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     PageRequest.*of*(0,2)  
     )  
     ).thenReturn(new PageImpl(new ArrayList<Referencedocument>()));  
     Page<Referencedocument> data = buildService.listDocReadyToBuild(0,2);  
     *assertThat*(data.isEmpty()).isTrue();  
     *assertThat*(data.getContent()).isEmpty();  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenGetAListOfDocReadyToBuild() {  
     *when*(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     PageRequest.*of*(1,1)  
     )  
     ).thenThrow(new MockitoException("Error while getting data"));  
     Throwable throwable = *assertThrows*(  
     Exception.class,  
     ()->buildService.listDocReadyToBuild(10,0)  
     );  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenGetDocumentInformationById() {  
     Optional<Object> objectOptional = Optional.*empty*();  
     *when*(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     Throwable throwable = *assertThrows*(  
     Exception.class,  
     ()->buildService.getDocumentInformationById(1L)  
     );  
     *assertThat*(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     private Cni getReferenceDocument() {  
     return new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     }  
      
     @Test  
     void itShouldGetDocumentInformationById() {  
     Optional<Object> objectOptional = Optional.*of*(this.getReferenceDocument());  
     *when*(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     Cni cni = (Cni) buildService.getDocumentInformationById(1L);  
     *assertThat*(cni.getIdDocumentReference()).isEqualTo(this.getReferenceDocument().getIdDocumentReference());  
     }  
      
     @Test  
     void itShouldReturnPageListOfDocRealTimeBuilding() {  
     *when*(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     PageRequest.*of*(0,2)  
     )  
     ).thenReturn(this.listDocInRealTimeBuildData());  
     Page<Referencedocument> data = buildService.listDocInRealTimeBuilding(0,2);  
     *verify*(referenceDocumentRepository, *times*(1)).findAll(  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     PageRequest.*of*(0,2)  
     );  
     *assertThat*(data.isEmpty()).isFalse();  
     *assertThat*(data.getContent()).isNotEmpty();  
     }  
      
     @Test  
     void itShouldReturnAnEmptyPageListOfDocRealTimeBuilding() {  
     *when*(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.BUILD,  
     StatusInTreatment.Waiting,  
     PageRequest.of(0,2)  
     )  
     ).thenReturn(new PageImpl(new ArrayList<Referencedocument>()));  
     Page<Referencedocument> data = buildService.listDocInRealTimeBuilding(0,2);  
     assertThat(data.isEmpty()).isTrue();  
     assertThat(data.getContent()).isEmpty();  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenGetAListOfDocRealTimeBuilding() {  
     when(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.BUILD,  
     StatusInTreatment.Waiting,  
     PageRequest.of(1,1)  
     )  
     ).thenThrow(new MockitoException("Error while getting data"));  
     assertThrows(  
     Exception.class,  
     ()->buildService.listDocInRealTimeBuilding(10,0)  
     );  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhileChangeStatusOfDocument() {  
     Optional<Object> objectOptional = Optional.empty();  
     when(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->buildService.changeStatusOfDocument(1L, StatusInTreatment.Ready)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     @Test  
     void itShouldChangeStatusOfDocument() {  
     Optional<Object> objectOptional = Optional.of(this.getReferenceDocument());  
     when(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     buildService.changeStatusOfDocument(1L, StatusInTreatment.Ready);  
     verify(referenceDocumentRepository, times(1)).save(any(Referencedocument.class));  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhileChangeStatusOfDocumentToEmit() {  
     Optional<Object> objectOptional = Optional.empty();  
     when(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->buildService.changeStatusOfDocumentToEmit(1L)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhileChangeStatusOfDocumentToEmit1() {  
     Optional<Object> objectOptional = Optional.of(this.getReferenceDocument());  
     when(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->buildService.changeStatusOfDocumentToEmit(1L)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Impossible d'effectuer cette operation");  
     }  
      
     @Test  
     void itShouldChangeStatusOfDocumentToEmit() {  
     Cni cni = this.getReferenceDocument();  
     cni.setStatusInTreatment(StatusInTreatment.Done);  
     cni.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.BUILD);  
     Optional<Object> objectOptional = Optional.of(cni);  
     when(referenceDocumentRepository.findOptionalReferenceDocumentById(1L)).thenReturn(objectOptional);  
     buildService.changeStatusOfDocumentToEmit(1L);  
     verify(referenceDocumentRepository, times(1)).save(any(Referencedocument.class));  
     }  
    }
12. package com.fabrication.emit.services;  
      
    import com.fabrication.agent.repositories.ReferenceDocumentRepository;  
    import com.fabrication.entities.\*;  
    import com.fabrication.services.EmailService;  
    import com.fabrication.services.EmailServiceImpl;  
    import com.fabrication.utils.Gender;  
    import com.fabrication.utils.PersonStatus;  
    import com.fabrication.utils.StatusInTreatment;  
    import com.fabrication.utils.StatusTreatmentSystemeList;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.mockito.exceptions.base.MockitoException;  
    import org.springframework.data.domain.Page;  
    import org.springframework.data.domain.PageImpl;  
    import org.springframework.data.domain.PageRequest;  
      
    import java.time.Instant;  
    import java.util.ArrayList;  
    import java.util.Date;  
    import java.util.List;  
    import java.util.Optional;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.junit.jupiter.api.Assertions.\*;  
    import static org.mockito.Mockito.\*;  
    import static org.mockito.Mockito.*times*;  
      
    class EmitServiceImplTest {  
      
     private ReferenceDocumentRepository referenceDocumentRepository;  
     private EmailService emailService;  
     private EmitService emitService;  
      
     @BeforeEach  
     void setUp(){  
     referenceDocumentRepository = *mock*(ReferenceDocumentRepository.class);  
     emailService = *mock*(EmailServiceImpl.class);  
     emitService = new EmitServiceImpl(referenceDocumentRepository,emailService);  
     }  
      
     private Imageadditionaldocument getImageAdditionalDocument(){  
     return new Imageadditionaldocument(  
     1L,  
     "jhvjhvjh",  
     Date.*from*(Instant.*now*()),  
     new Agent(),  
     new Client(),  
     new Cni()  
     );  
     }  
      
     private Page<Referencedocument> listDocReadyToBuildData(){  
     Cni cni = getReferenceDocument();  
     Cni cni1 = new Cni(  
     89L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     Cni cni2 = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     198L,  
     "kjjlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     Cni cni3 = new Cni(  
     889L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     7L,  
     "jlkfsf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     List<Referencedocument> referencedocumentList = new ArrayList<>();  
     referencedocumentList.add(cni);  
     referencedocumentList.add(cni1);  
     referencedocumentList.add(cni2);  
     referencedocumentList.add(cni3);  
      
     return new PageImpl<Referencedocument>(referencedocumentList, PageRequest.*of*(0, 2),4L);  
      
     }  
      
     private Page<Referencedocument> listDocInRealTimeBuildData(){  
     Cni cni = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni1 = new Cni(  
     89L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni2 = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     198L,  
     "kjjlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni3 = new Cni(  
     889L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     7L,  
     "jlkfsf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*EMIT*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     List<Referencedocument> referencedocumentList = new ArrayList<>();  
     referencedocumentList.add(cni);  
     referencedocumentList.add(cni1);  
     referencedocumentList.add(cni2);  
     referencedocumentList.add(cni3);  
      
     return new PageImpl<Referencedocument>(referencedocumentList, PageRequest.*of*(0, 2),4L);  
      
     }  
     private Cni getReferenceDocument() {  
     return new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.ACTIVE,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.ACTIVE,  
     "123456",  
     Date.from(Instant.now())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.EMIT,  
     StatusInTreatment.Ready,  
     null  
     );  
     }  
      
     @Test  
     void itShouldReturnAPageListOfDocumentToEmit() {  
     when(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.EMIT,  
     StatusInTreatment.Ready,  
     PageRequest.of(0,2)  
     )  
     ).thenReturn(this.listDocReadyToBuildData());  
     Page<Object> data = emitService.listDocumentToEmit(0,2);  
     verify(referenceDocumentRepository, times(1)).findAll(  
     StatusTreatmentSystemeList.EMIT,  
     StatusInTreatment.Ready,  
     PageRequest.of(0,2)  
     );  
     assertThat(data.isEmpty()).isFalse();  
     assertThat(data.getContent()).isNotEmpty();  
     }  
      
     @Test  
     void itShouldReturnEmptyPageOfDocumentToEmit(){  
     when(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.EMIT,  
     StatusInTreatment.Ready,  
     PageRequest.of(0,2)  
     )  
     ).thenReturn(new PageImpl(new ArrayList<Referencedocument>()));  
     Page<Object> data = emitService.listDocumentToEmit(0,2);  
     assertThat(data.isEmpty()).isTrue();  
     assertThat(data.getContent()).isEmpty();  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenGetAListOfDocReadyToEmit() {  
     when(referenceDocumentRepository  
     .findAll(  
     StatusTreatmentSystemeList.BUILD,  
     StatusInTreatment.Ready,  
     PageRequest.of(1,1)  
     )  
     ).thenThrow(new MockitoException("Error while getting data"));  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->emitService.listDocumentToEmit(10,0)  
     );  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenGetDocumentInformationById() {  
     Optional<Object> objectOptional = Optional.empty();  
     when(referenceDocumentRepository  
     .getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)  
     ).thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->emitService.getReferenceDocument(1L)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     @Test  
     void itShouldGetDocumentInformationById() {  
     Optional<Object> objectOptional = Optional.of(this.getReferenceDocument());  
     when(referenceDocumentRepository  
     .getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)  
     )  
     .thenReturn(objectOptional);  
     Cni cni = (Cni) emitService.getReferenceDocument(1L);  
     assertThat(cni.getIdDocumentReference()).isEqualTo(this.getReferenceDocument().getIdDocumentReference());  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenGetDocumentInformationByIdStatusIsNotEmit() {  
     Cni cni = this.getReferenceDocument();  
     cni.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.VALIDATE);  
     Optional<Object> objectOptional = Optional.of(cni);  
     when(referenceDocumentRepository  
     .getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)  
     ).thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->emitService.getReferenceDocument(1L)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     @Test  
     void itShouldSendNotificationEmailToClient() {  
     Optional<Object> objectOptional = Optional.of(this.getReferenceDocument());  
     String code = "123654";  
     when(referenceDocumentRepository  
     .getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)  
     ).thenReturn(objectOptional);  
     when(emailService.generateValidationCode()).thenReturn(code);  
     emitService.sendEmailToNotifiedClient(1L);  
     verify(emailService, times(1)).generateValidationCode();  
     verify(referenceDocumentRepository, times(1)).save(any(Referencedocument.class));  
     }  
      
     @Test  
     void itShouldThrowAnExceptionWhenSendNotificationEmailToClient() {  
     Cni cni = this.getReferenceDocument();  
     cni.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.VALIDATE);  
     Optional<Object> objectOptional = Optional.of(cni);  
     when(referenceDocumentRepository  
     .getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)  
     )  
     .thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->emitService.sendEmailToNotifiedClient(1L)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     @Test  
     void itShouldThrowAnExceptionNullPointerWhenSendNotificationEmailToClient() {  
     Optional<Object> objectOptional = Optional.empty();  
     when(referenceDocumentRepository  
     .getCurrentReferenceDocumentByClientId(1L, StatusTreatmentSystemeList.FINISH)  
     ).thenReturn(objectOptional);  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->emitService.sendEmailToNotifiedClient(1L)  
     );  
     assertThat(throwable.getMessage()).isEqualTo("Unexisting Element");  
     }  
      
     @Test  
     void verifiedReferenceDocumentToReturnDocumentToClient() {  
     }  
    }
13. package com.fabrication.exceptions;  
      
    import com.fabrication.utils.ErrorBody;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.springframework.http.HttpStatus;  
    import org.springframework.http.ResponseEntity;  
    import org.springframework.mock.web.MockHttpServletRequest;  
    import org.springframework.web.context.request.ServletWebRequest;  
      
    import java.util.Objects;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
      
    class GeneralExceptionTreatmentTest {  
      
     private GeneralExceptionTreatement generalExceptionTreatement;  
      
     @BeforeEach  
     void setUp() {  
     this.generalExceptionTreatement = new GeneralExceptionTreatement();  
     }  
      
     @Test  
     void customExceptionController() {  
      
     ErrorBody errorBody = new ErrorBody(  
     "Exception",  
     HttpStatus.*NOT\_FOUND*.toString(),  
     "/api/v1/data"  
     );  
      
     ResponseEntity<?> responseEntity = generalExceptionTreatement.customExceptionController(  
     new RuntimeException("Exception"),  
     new ServletWebRequest(  
     new MockHttpServletRequest("get","/api/v1/data")  
     ));  
     assertCustomException(errorBody, responseEntity);  
     }  
      
     private void assertCustomException(ErrorBody errorBody, ResponseEntity<?> responseEntity) {  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*NOT\_FOUND*);  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualByComparingTo(404);  
     ErrorBody errorReturn = (ErrorBody) responseEntity.getBody();  
     *assertThat*(Objects.*requireNonNull*(responseEntity.getBody()).getClass()).isEqualTo(errorBody.getClass());  
     assert errorReturn != null;  
     *assertThat*(errorReturn.getCode()).isEqualTo(errorBody.getCode());  
     *assertThat*(errorReturn.getResource()).isEqualTo(errorBody.getResource());  
     }  
    }
14. Tests Under controllers
15. package com.fabrication.agent.controllers;  
      
    import static org.junit.jupiter.api.Assertions.*assertEquals*;  
    import static org.junit.jupiter.api.Assertions.*assertTrue*;  
    import static org.mockito.Mockito.*any*;  
    import static org.mockito.Mockito.*mock*;  
    import static org.mockito.Mockito.*verify*;  
    import static org.mockito.Mockito.*when*;  
      
    import com.fabrication.agent.repositories.AdditionalDocumentsRepository;  
    import com.fabrication.agent.repositories.FileRepository;  
    import com.fabrication.agent.repositories.ReferenceDocumentRepository;  
    import com.fabrication.agent.services.FileResourceService;  
    import com.fabrication.agent.services.FileResourceServiceImpl;  
    import com.fabrication.client.repositories.PersonRepository;  
    import com.fabrication.client.services.PersonServiceImpl;  
    import com.fabrication.services.EmailServiceImpl;  
      
    import java.io.ByteArrayInputStream;  
    import java.io.IOException;  
    import java.util.HashMap;  
    import java.util.Map;  
      
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Disabled;  
    import org.junit.jupiter.api.Test;  
    import org.springframework.http.HttpStatus;  
    import org.springframework.http.ResponseEntity;  
    import org.springframework.mail.javamail.JavaMailSenderImpl;  
    import org.springframework.mock.web.MockMultipartFile;  
    import org.springframework.web.multipart.MultipartFile;  
      
    class FileControllerTest {  
      
     private FileController fileController;  
     private FileResourceService fileResourceService;  
      
     @BeforeEach  
     void setUp(){  
     fileResourceService = *mock*(FileResourceServiceImpl.class);  
     fileController = new FileController(fileResourceService);  
     }  
      
      
     @Test  
     void itShouldUploadABirthCertificate() throws Exception {  
     *when*(fileResourceService.uploadBirthCertificate((MultipartFile) *any*(), (Long) *any*(), (Long) *any*(), (String) *any*()))  
     .thenReturn("Birth Certificate");  
     ResponseEntity<?> actualBirthCertificateResult = fileController.uploadBirthCertificate(  
     new MockMultipartFile("Name", new ByteArrayInputStream("AAAAAAAA".getBytes("UTF-8"))), 123L , 123L,"String");  
     Map<String, String> data = new HashMap<>();  
     data.put("response", "Birth Certificate");  
     *assertEquals*(data, actualBirthCertificateResult.getBody());  
     *assertEquals*(HttpStatus.*OK*, actualBirthCertificateResult.getStatusCode());  
     *assertTrue*(actualBirthCertificateResult.getHeaders().isEmpty());  
     *verify*(fileResourceService).uploadBirthCertificate((MultipartFile) *any*(), (Long) *any*(), (Long) *any*(), (String) *any*());  
     }  
      
      
      
     @Test  
     void itShouldUploadANationalCertificate() throws Exception {  
     *when*(fileResourceService.uploadNationalCertificate((MultipartFile) *any*(), (Long) *any*(), (Long) *any*(), (String) *any*()))  
     .thenReturn("Upload national certificate Successful");  
     ResponseEntity<?> actualNationalCertificateResult = fileController.uploadNationalCertificate(  
     new MockMultipartFile(  
     "Name",  
     new ByteArrayInputStream(  
     "AAAAAAAA".getBytes("UTF-8")  
     )  
     ),  
     123L,  
     123L,  
     "(String) any()"  
     );  
     Map<String, String> data = new HashMap<>();  
     data.put("response", "Upload national certificate Successful");  
     *assertEquals*(data, actualNationalCertificateResult.getBody());  
     *assertEquals*(HttpStatus.*OK*, actualNationalCertificateResult.getStatusCode());  
     *assertTrue*(actualNationalCertificateResult.getHeaders().isEmpty());  
     *verify*(fileResourceService).uploadNationalCertificate((MultipartFile) *any*(), (Long) *any*(),  
     (Long) *any*(), (String) *any*());  
     }  
      
      
      
     @Test  
     void itShouldUploadALostCertificate() throws Exception {  
     *when*(fileResourceService.uploadLostCertificate((MultipartFile) *any*(), (Long) *any*(), (Long) *any*(), (String) *any*()))  
     .thenReturn("Upload lostcertificate Successful");  
     ResponseEntity<?> actualLostCertificateResult = fileController.uploadLostCertificate(  
     new MockMultipartFile("Name", new ByteArrayInputStream("AAAAAAAA".getBytes("UTF-8"))), 123L, 123L, "(String) any()");  
     Map<String, String> data = new HashMap<>();  
     data.put("response", "Upload lostcertificate Successful");  
     *assertEquals*(data, actualLostCertificateResult.getBody());  
     *assertEquals*(HttpStatus.*OK*, actualLostCertificateResult.getStatusCode());  
     *assertTrue*(actualLostCertificateResult.getHeaders().isEmpty());  
     *verify*(fileResourceService).uploadLostCertificate((MultipartFile) any(), (Long) any(), (Long) any(), (String) any());  
     }  
      
      
      
     @Test  
     void itShouldUploadAnImage() throws Exception {  
     when(fileResourceService.uploadImage((String) any(), (Long) any(), (Long) any(), (String) any()))  
     .thenReturn("Upload Image Successful");  
     ResponseEntity<?> actualImageResult = fileController.uploadImage(  
     "AAAAAAAA",  
     123L,  
     12L,  
     "(String) any()");  
     Map<String, String> data = new HashMap<>();  
     data.put("response", "Upload Image Successful");  
     assertEquals(data, actualImageResult.getBody());  
     assertEquals(HttpStatus.OK, actualImageResult.getStatusCode());  
     assertTrue(actualImageResult.getHeaders().isEmpty());  
     verify(fileResourceService).uploadImage((String) any(), any(), (Long) any(), (String) any());  
     }  
    }
16. package com.fabrication.agent.controllers;  
      
    import com.fabrication.agent.services.ReferentDocumentService;  
    import com.fabrication.entities.Additionaldocument;  
    import com.fabrication.entities.Cni;  
    import com.fabrication.entities.Passport;  
    import com.fabrication.entities.Referencedocument;  
    import com.fabrication.utils.Gender;  
    import com.fabrication.utils.StatusInTreatment;  
    import com.fabrication.utils.StatusTreatmentSystemeList;  
    import com.fasterxml.jackson.databind.ObjectMapper;  
      
    import java.time.LocalDate;  
    import java.time.LocalDateTime;  
    import java.time.ZoneId;  
      
    import java.util.ArrayList;  
    import java.util.Date;  
    import java.util.List;  
      
    import org.junit.jupiter.api.Test;  
    import org.junit.jupiter.api.extension.ExtendWith;  
    import org.springframework.beans.factory.annotation.Autowired;  
    import org.springframework.boot.test.mock.mockito.MockBean;  
    import org.springframework.data.domain.Page;  
    import org.springframework.data.domain.PageImpl;  
    import org.springframework.data.domain.PageRequest;  
    import org.springframework.http.HttpStatus;  
    import org.springframework.http.MediaType;  
    import org.springframework.http.ResponseEntity;  
    import org.springframework.test.context.ContextConfiguration;  
    import org.springframework.test.context.junit.jupiter.SpringExtension;  
    import org.springframework.test.web.servlet.MockMvc;  
    import org.springframework.test.web.servlet.ResultActions;  
    import org.springframework.test.web.servlet.request.MockHttpServletRequestBuilder;  
    import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
    import org.springframework.test.web.servlet.result.MockMvcResultMatchers;  
    import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.mockito.Mockito.\*;  
      
    @ContextConfiguration(classes = {ReferenceDocumentController.class})  
    @ExtendWith(SpringExtension.class)  
    class ReferenceDocumentControllerTest {  
     @Autowired  
     private ReferenceDocumentController referenceDoController;  
      
     @MockBean  
     private ReferentDocumentService referentDocumentService;  
      
     private Cni getCni() {  
     Cni cni = new Cni();  
     cni.setAddress("add");  
     LocalDateTime atStartOfDayResult = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     cni.setDateOfBirth(Date.*from*(atStartOfDayResult.atZone(ZoneId.*of*("UTC")).toInstant()));  
     LocalDateTime atStartOfDayResult1 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     cni.setDeliveryDate(Date.*from*(atStartOfDayResult1.atZone(ZoneId.*of*("UTC")).toInstant()));  
     cni.setDocumentNumber("42");  
     LocalDateTime atStartOfDayResult2 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     cni.setExpirationDate(Date.*from*(atStartOfDayResult2.atZone(ZoneId.*of*("UTC")).toInstant()));  
     cni.setFirstName("fn");  
     cni.setGender(Gender.*FEMALE*);  
     cni.setIdDocumentReference(5L);  
     cni.setLastName("ln");  
     cni.setNameOfFather("nf");  
     cni.setNameOfMother("nm");  
     cni.setPosteIdentification("pi");  
     cni.setProfession("p");  
     cni.setReferenceNumber("42");  
     cni.setStatusInTreatment(StatusInTreatment.*Ready*);  
     cni.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     LocalDateTime atStartOfDayResult3 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     cni.setWithdrawalDate(Date.*from*(atStartOfDayResult3.atZone(ZoneId.*of*("UTC")).toInstant()));  
     return cni;  
     }  
     private Cni getCni1() {  
     Cni cni = new Cni();  
     ArrayList<Additionaldocument> additionalDocumentList = new ArrayList<>();  
     cni.setAddress("add1");  
     ZoneId zone = ZoneId.*of*("UTC");  
     cni.setDateOfBirth(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone).toInstant()));  
     ZoneId zone1 = ZoneId.*of*("UTC");  
     cni.setDeliveryDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone1).toInstant()));  
     cni.setDocumentNumber("42");  
     ZoneId zone2 = ZoneId.*of*("UTC");  
     cni.setExpirationDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone2).toInstant()));  
     cni.setFirstName("FN");  
     cni.setGender(Gender.*FEMALE*);  
     cni.setIdDocumentReference(5L);  
     cni.setLastName("LN");  
     cni.setNameOfFather("FN");  
     cni.setNameOfMother("MN");  
     cni.setPosteIdentification("Pi");  
     cni.setProfession("Profession");  
     cni.setReferenceNumber("42");  
     cni.setStatusInTreatment(StatusInTreatment.*Ready*);  
     cni.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     return cni;  
     }  
      
     private Passport getPassport() {  
     Passport passport = new Passport();  
     ArrayList<Additionaldocument> additionalDocumentList = new ArrayList<>();  
     passport.setAddress("PA");  
     passport.setCountry("GB");  
     LocalDateTime atStartOfDayResult = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setDateOfBirth(Date.*from*(atStartOfDayResult.atZone(ZoneId.*of*("UTC")).toInstant()));  
     LocalDateTime atStartOfDayResult1 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setDeliveryDate(Date.*from*(atStartOfDayResult1.atZone(ZoneId.*of*("UTC")).toInstant()));  
     passport.setDocumentNumber("44");  
     LocalDateTime atStartOfDayResult2 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setExpirationDate(Date.*from*(atStartOfDayResult2.atZone(ZoneId.*of*("UTC")).toInstant()));  
     passport.setFirstName("fn");  
     passport.setGender(Gender.*FEMALE*);  
     passport.setIdDocumentReference(6L);  
     passport.setLastName("ln");  
     passport.setNameOfFather("Nf");  
     passport.setNameOfMother("Nm");  
     passport.setProfession("p");  
     passport.setReferenceNumber("44");  
     passport.setStatusInTreatment(StatusInTreatment.*Ready*);  
     passport.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     LocalDateTime atStartOfDayResult3 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setWithdrawalDate(Date.*from*(atStartOfDayResult3.atZone(ZoneId.*of*("UTC")).toInstant()));  
     return passport;  
     }  
     private Passport getPassportupd() {  
     Passport passport = new Passport();  
     ArrayList<Additionaldocument> additionalDocumentList = new ArrayList<>();  
     passport.setAddress("PA");  
     passport.setCountry("GB");  
     LocalDateTime atStartOfDayResult = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setDateOfBirth(Date.*from*(atStartOfDayResult.atZone(ZoneId.*of*("UTC")).toInstant()));  
     LocalDateTime atStartOfDayResult1 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setDeliveryDate(Date.*from*(atStartOfDayResult1.atZone(ZoneId.*of*("UTC")).toInstant()));  
     passport.setDocumentNumber("44");  
     ZoneId zone2 = ZoneId.*of*("UTC");  
     LocalDateTime atStartOfDayResult2 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     passport.setExpirationDate(Date.*from*(atStartOfDayResult2.atZone(ZoneId.*of*("UTC")).toInstant()));  
     passport.setGender(Gender.*FEMALE*);  
     passport.setIdDocumentReference(6l);  
     passport.setLastName("ln");  
     passport.setNameOfFather("Nf");  
     passport.setNameOfMother("Nm");  
     passport.setProfession("p");  
     passport.setReferenceNumber("44");  
     passport.setStatusInTreatment(StatusInTreatment.*Ready*);  
     passport.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     return passport;  
     }  
      
      
     @Test  
     void itShouldSaveCni() throws Exception {  
      
     Cni cni1 = new Cni();  
     MockHttpServletRequestBuilder contentTypeResult = MockMvcRequestBuilders  
     .*post*("/api/v1/document/cni", cni1)  
     .contentType(MediaType.*APPLICATION\_JSON*);  
      
     Cni cni = new Cni();  
     cni.setAddress("42 Main St");  
     ZoneId zone = ZoneId.*of*("UTC");  
     cni.setDateOfBirth(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone).toInstant()));  
     ZoneId zone1 = ZoneId.*of*("UTC");  
     cni.setDeliveryDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone1).toInstant()));  
     cni.setDocumentNumber("42");  
     ZoneId zone2 = ZoneId.*of*("UTC");  
     cni.setExpirationDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone2).toInstant()));  
     cni.setFirstName("Jane");  
     cni.setGender(Gender.*FEMALE*);  
     cni.setIdDocumentReference(5L);  
     cni.setLastName("Doe");  
     cni.setNameOfFather("Name Of Father");  
     cni.setNameOfMother("Name Of Mother");  
     cni.setPosteIdentification("Poste Identification");  
     cni.setProfession("Profession");  
     cni.setReferenceNumber("42");  
     cni.setStatusInTreatment(StatusInTreatment.*Ready*);  
     cni.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     ZoneId zone3 = ZoneId.*of*("UTC");  
     cni.setWithdrawalDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone3).toInstant()));  
     ObjectMapper objectMapper = new ObjectMapper();  
     MockHttpServletRequestBuilder requestBuilder = contentTypeResult.content(objectMapper.writeValueAsString(cni));  
     Object[] controllers = new Object[]{referenceDoController};  
     MockMvc buildResult = MockMvcBuilders.*standaloneSetup*(controllers).build();  
      
     ResultActions actualPerformResult = buildResult.perform(requestBuilder);  
      
     }  
      
     @Test  
     void itShouldFetchedCni() throws Exception {  
     *when*(referentDocumentService.getAllDocuments()).thenReturn(new ArrayList<>());  
     MockHttpServletRequestBuilder requestBuilder = MockMvcRequestBuilders.*get*("/api/v1/document/find\_All\_Documents");  
     MockMvcBuilders.*standaloneSetup*(referenceDoController)  
     .build()  
     .perform(requestBuilder)  
     .andExpect(MockMvcResultMatchers.*status*().isOk())  
     .andExpect(MockMvcResultMatchers.*content*().contentType("application/json"))  
     .andExpect(MockMvcResultMatchers.*content*().string("[]"));  
     }  
      
     @Test  
     void itShouldUpdateCni() throws Exception {  
      
     Cni cni = getCni();  
     *when*(referentDocumentService.updateCni((Cni) *any*(), (Long) *any*())).thenReturn(cni);  
      
     Cni cni1 = getCni1();  
     LocalDateTime atStartOfDayResult7 = LocalDate.*of*(1970, 1, 1).atStartOfDay();  
     cni1.setWithdrawalDate(Date.*from*(atStartOfDayResult7.atZone(ZoneId.*of*("UTC")).toInstant()));  
     String content = (new ObjectMapper()).writeValueAsString(cni1);  
     MockHttpServletRequestBuilder requestBuilder = MockMvcRequestBuilders.*put*("/api/v1/document/cni/{id}", 5L)  
     .contentType(MediaType.*APPLICATION\_JSON*)  
     .content(content);  
     MockMvcBuilders.*standaloneSetup*(referenceDoController)  
     .build()  
     .perform(requestBuilder)  
     .andExpect(MockMvcResultMatchers.*status*().isCreated())  
     .andExpect(MockMvcResultMatchers.*content*().contentType("application/json"))  
     .andExpect(MockMvcResultMatchers.*content*()  
     .string(  
     "{\"idDocumentReference\":5,\"documentNumber\":\"42\",\"referenceNumber\":\"42\",\"lastName\":\"ln\",\"firstName\":"  
     + "\"fn\",\"dateOfBirth\":0,\"gender\":\"FEMALE\",\"profession\":\"p\",\"nameOfFather\":\"nf\","  
     + "\"nameOfMother\":\"nm\",\"deliveryDate\":0,\"expirationDate\":0,\"address\":\"add\","  
     + "\"agent\":null,\"client\":null,\"imageadditionaldocument\":null,\"statusTreatmentSystemeList\""  
     + ":\"FORM\",\"statusInTreatment\":\"Ready\",\"withdrawalDate\":0,\"birthcertificate\":null,"  
     + "\"nationalitycertificate\":null,\"lostcertificate\":null,\"biometric\":null,\"posteIdentification\":\"pi\"}"));  
      
      
       
      
     }  
      
     @Test  
     void itShouldDeleteCniById() throws Exception {  
     *doNothing*().when(referentDocumentService).deleteCni((Long) *any*());  
     MockHttpServletRequestBuilder requestBuilder = MockMvcRequestBuilders.*delete*("/api/v1/document/cni/{id}", 5L);  
     MockMvcBuilders.*standaloneSetup*(referenceDoController)  
     .build()  
     .perform(requestBuilder)  
     .andExpect(MockMvcResultMatchers.*status*().isOk())  
     .andExpect(MockMvcResultMatchers.*content*().contentType("text/plain;charset=ISO-8859-1"))  
     .andExpect(MockMvcResultMatchers.*content*().string("Deleted Successfully"));  
     }  
      
      
    ///Passports  
      
     @Test  
     void itShouldSavePassport() throws Exception {  
      
     Passport passport1 = new Passport();  
     MockHttpServletRequestBuilder contentTypeResult = MockMvcRequestBuilders  
     .*post*("/api/v1/document/passport", passport1)  
     .contentType(MediaType.*APPLICATION\_JSON*);  
      
     Passport passport = new Passport();  
     ArrayList<Additionaldocument> additionalDocumentList = new ArrayList<>();  
     passport.setAddress("42 Main St");  
     passport.setCountry("GB");  
     ZoneId zone = ZoneId.*of*("UTC");  
     passport.setDateOfBirth(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone).toInstant()));  
     ZoneId zone1 = ZoneId.*of*("UTC");  
     passport.setDeliveryDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone1).toInstant()));  
     passport.setDocumentNumber("42");  
     ZoneId zone2 = ZoneId.*of*("UTC");  
     passport.setExpirationDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone2).toInstant()));  
     passport.setFirstName("Jane");  
     passport.setGender(Gender.*FEMALE*);  
     passport.setIdDocumentReference(6L);  
     passport.setLastName("Doe");  
     passport.setNameOfFather("Name Of Father");  
     passport.setNameOfMother("Name Of Mother");  
     passport.setProfession("Profession");  
     passport.setReferenceNumber("42");  
     passport.setStatusInTreatment(StatusInTreatment.*Ready*);  
     passport.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     ZoneId zone3 = ZoneId.*of*("UTC");  
     passport.setWithdrawalDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone3).toInstant()));  
      
     ObjectMapper objectMapper = new ObjectMapper();  
     MockHttpServletRequestBuilder requestBuilder = contentTypeResult  
     .content(objectMapper.writeValueAsString(passport));  
     Object[] controllers = new Object[]{referenceDoController};  
     MockMvc buildResult = MockMvcBuilders.*standaloneSetup*(controllers).build();  
      
      
     ResultActions actualPerformResult = buildResult.perform(requestBuilder);  
      
     }  
      
      
      
      
     @Test  
     void itShouldUpdatePassPort() throws Exception {  
      
     Passport passport1 = new Passport();  
     MockHttpServletRequestBuilder contentTypeResult = MockMvcRequestBuilders  
     .*put*("/api/v1/document/passport/{id}", passport1)  
     .contentType(MediaType.*APPLICATION\_JSON*);  
      
     Passport passport = new Passport();  
     ArrayList<Additionaldocument> additionalDocumentList = new ArrayList<>();  
     passport.setAddress("42 Main St");  
     passport.setCountry("GB");  
     ZoneId zone = ZoneId.*of*("UTC");  
     passport.setDateOfBirth(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone).toInstant()));  
     ZoneId zone1 = ZoneId.*of*("UTC");  
     passport.setDeliveryDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone1).toInstant()));  
     passport.setDocumentNumber("42");  
     ZoneId zone2 = ZoneId.*of*("UTC");  
     passport.setExpirationDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone2).toInstant()));  
     passport.setFirstName("Jane");  
     passport.setGender(Gender.*FEMALE*);  
     passport.setIdDocumentReference(6L);  
     passport.setLastName("Doe");  
     passport.setNameOfFather("Name Of Father");  
     passport.setNameOfMother("Name Of Mother");  
     passport.setProfession("Profession");  
     passport.setReferenceNumber("42");  
     passport.setStatusInTreatment(StatusInTreatment.*Ready*);  
     passport.setStatusTreatmentSystemeList(StatusTreatmentSystemeList.*FORM*);  
     ZoneId zone3 = ZoneId.*of*("UTC");  
     passport.setWithdrawalDate(Date.*from*(LocalDate.*of*(1970, 1, 1).atStartOfDay().atZone(zone3).toInstant()));  
      
     ObjectMapper objectMapper = new ObjectMapper();  
     MockHttpServletRequestBuilder requestBuilder = contentTypeResult  
     .content(objectMapper.writeValueAsString(passport));  
     Object[] controllers = new Object[]{referenceDoController};  
     MockMvc buildResult = MockMvcBuilders.*standaloneSetup*(controllers).build();  
      
      
     ResultActions actualPerformResult = buildResult.perform(requestBuilder);  
      
     }  
      
     @Test  
     void itShouldInitCni(){  
     ResponseEntity responseEntity = referenceDoController.initCni(1L);  
     *verify*(referentDocumentService,*times*(1)).initCni(1L);  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(204);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*NO\_CONTENT*);  
     }  
      
     @Test  
     void itShouldInitPassPort(){  
     ResponseEntity responseEntity = referenceDoController.initPassPort(1L);  
     *verify*(referentDocumentService,*times*(1)).initPassPort(1L);  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(204);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*NO\_CONTENT*);  
     }  
      
     @Test  
     void itShouldFindAlldDocument(){  
     List<Referencedocument> referenceDocumentList = new ArrayList<>();  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     Page<Referencedocument> referenceDocumentPageMock = new PageImpl<>(  
     referenceDocumentList, PageRequest.*of*(0,5),5  
     );  
     StatusTreatmentSystemeList statusTreatmentSystemeList = StatusTreatmentSystemeList.*BUILD*;  
      
     //When  
     *when*(  
     referentDocumentService.findAllReferenceByStatusTreatmentSystemListAndPage(  
     statusTreatmentSystemeList,  
     0,  
     5)  
     )  
     .thenReturn(referenceDocumentPageMock);  
     ResponseEntity responseEntity = referenceDoController.findAllDocumentByPage(  
     0,  
     5,  
     statusTreatmentSystemeList  
     );  
     *verify*(referentDocumentService,*times*(1))  
     .findAllReferenceByStatusTreatmentSystemListAndPage(  
     StatusTreatmentSystemeList.*BUILD*,  
     0,  
     5);  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*OK*);  
     *assertThat*(responseEntity.getBody()).isNotNull();  
     }  
      
     @Test  
     void itShouldFindAllBuildDocument(){  
     List<Referencedocument> referenceDocumentList = new ArrayList<>();  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     referenceDocumentList.add(this.getCni1());  
     referenceDocumentList.add(this.getCni());  
     Page<Referencedocument> referenceDocumentPageMock = new PageImpl<>(  
     referenceDocumentList, PageRequest.*of*(0,5),5  
     );  
     StatusInTreatment statusInTreatment = StatusInTreatment.*Waiting*;  
      
     //When  
     *when*(  
     referentDocumentService.findAllReferenceInBuildingStepByStatusInTreatmentAndPage(  
     statusInTreatment,  
     0,  
     5)  
     )  
     .thenReturn(referenceDocumentPageMock);  
     ResponseEntity responseEntity = referenceDoController.findAllDocumentBuildByPage(  
     0,  
     5,  
     statusInTreatment  
     );  
     *verify*(referentDocumentService,*times*(1))  
     .findAllReferenceInBuildingStepByStatusInTreatmentAndPage(  
     statusInTreatment,  
     0,  
     5);  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*OK*);  
     *assertThat*(responseEntity.getBody()).isNotNull();  
     }  
    }
17. package com.fabrication.client.controllers;  
      
    import com.fabrication.client.services.PersonService;  
    import com.fabrication.client.services.PersonServiceImpl;  
    import com.fabrication.utils.LoginBean;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.springframework.http.HttpStatus;  
    import org.springframework.http.ResponseEntity;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.junit.jupiter.api.Assertions.\*;  
    import static org.mockito.Mockito.\*;  
      
    class ClientRestControllerTest {  
      
     private ClientRestController clientRestController;  
     private PersonService personService;  
      
     @BeforeEach  
     void setUp(){  
     personService = *mock*(PersonServiceImpl.class);  
     clientRestController = new ClientRestController(personService);  
     }  
      
     @Test  
     void itShouldDisableClient() {  
     ResponseEntity<?> responseEntity = clientRestController.disableUser("email");  
     *verify*(personService, *times*(1)).disableClient("email");  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(204);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*NO\_CONTENT*);  
     *assertThat*(responseEntity.getBody()).isNull();  
     }  
      
     @Test  
     void itShouldValidateClientCode() {  
     LoginBean loginBean = new LoginBean("login","123456");  
     ResponseEntity<?> responseEntity = clientRestController.validateClient(loginBean);  
     *verify*(personService, *times*(1)).codeClientValidation(loginBean.getLogin(),loginBean.getPassword());  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(204);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*NO\_CONTENT*);  
     *assertThat*(responseEntity.getBody()).isNull();  
     }  
    }
18. package com.fabrication.client.controllers;  
      
    import com.fabrication.client.services.PersonService;  
    import com.fabrication.client.services.PersonServiceImpl;  
    import com.fabrication.entities.Agent;  
    import com.fabrication.entities.Client;  
    import com.fabrication.exceptions.ResourceNotFoundException;  
    import com.fabrication.utils.LoginBean;  
    import com.fabrication.utils.PersonStatus;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.springframework.http.HttpStatus;  
    import org.springframework.http.ResponseEntity;  
      
    import java.time.Instant;  
    import java.util.Date;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.junit.jupiter.api.Assertions.*assertThrows*;  
    import static org.mockito.Mockito.*mock*;  
    import static org.mockito.Mockito.*when*;  
      
    class LoginControllerTest {  
      
     private LoginController loginController;  
     private PersonService personService;  
      
     @BeforeEach  
     void setUp(){  
     personService = *mock*(PersonServiceImpl.class);  
     loginController = new LoginController(personService);  
     }  
      
     @Test  
     void givenEmail\_itShouldConnectClient() {  
     //Given  
     String login = "tototo@gmail.com";  
     Client client = new Client(  
     1L,  
     "tototo@gmail.com",  
     PersonStatus.*ACTIVE*,  
     "012345",  
     Date.*from*(Instant.*now*())  
     );  
      
     //When  
     *when*(personService.connectClient(login)).thenReturn(client);  
      
     ResponseEntity<?> responseEntity = loginController.connectClient(login);  
      
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     *assertThat*(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.*OK*);  
     *assertThat*(responseEntity.getBody()).isEqualTo(client);  
     }  
      
     @Test  
     void givenNullEmail\_itShouldThrowAnExceptionToConnectClient() {  
     //When  
     *when*(personService.connectClient("email")).thenThrow(new ResourceNotFoundException("Email is null"));  
      
     Throwable exception = *assertThrows*(  
     ResourceNotFoundException.class,  
     ()-> loginController.connectClient("email")  
     );  
      
     //Then  
     assertThat(exception.getMessage()).isEqualTo("Email is null");  
     }  
      
     @Test  
     void givenLoginAndPassword\_itShouldConnectAgent() {  
     //Given  
     LoginBean loginBean = new LoginBean("login", "password");  
      
     //When  
     Agent agent = new Agent(  
     1L,  
     "tototo@gmail.com",  
     PersonStatus.ACTIVE,  
     "agentLastName",  
     "agentFirstName",  
     "login",  
     "password"  
     );  
     when(personService.connectAgent(loginBean)).thenReturn(agent);  
     ResponseEntity<?> responseEntity = loginController.connectAgent(loginBean);  
      
     //Then  
     assertThat(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     assertThat(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.OK);  
     assertThat(responseEntity.getBody()).isEqualTo(agent);  
     }  
      
     @Test  
     void givenLogin\_itShouldThrowAnExceptionToConnectAgent() {  
     //Given  
     LoginBean loginBean = new LoginBean("login", "password");  
      
     //When  
     when(personService.connectAgent(loginBean)).thenThrow(new ResourceNotFoundException("Login and/or Password invalid"));  
      
     Throwable exception = assertThrows(  
     ResourceNotFoundException.class,  
     ()-> loginController.connectAgent(loginBean)  
     );  
      
     //Then  
     assertThat(exception.getMessage()).isEqualTo("Login and/or Password invalid");  
     }  
    }
19. package com.fabrication.build.controller;  
      
    import com.fabrication.build.services.BuildService;  
    import com.fabrication.build.services.BuildServiceImpl;  
    import com.fabrication.entities.\*;  
    import com.fabrication.exceptions.ResourceNotFoundException;  
    import com.fabrication.utils.Gender;  
    import com.fabrication.utils.PersonStatus;  
    import com.fabrication.utils.StatusInTreatment;  
    import com.fabrication.utils.StatusTreatmentSystemeList;  
    import org.junit.jupiter.api.BeforeEach;  
    import org.junit.jupiter.api.Test;  
    import org.mockito.exceptions.base.MockitoException;  
    import org.springframework.data.domain.Page;  
    import org.springframework.data.domain.PageImpl;  
    import org.springframework.data.domain.PageRequest;  
    import org.springframework.http.HttpStatus;  
    import org.springframework.http.ResponseEntity;  
      
    import java.time.Instant;  
    import java.util.ArrayList;  
    import java.util.Date;  
    import java.util.List;  
    import java.util.Optional;  
      
    import static org.assertj.core.api.Assertions.*assertThat*;  
    import static org.junit.jupiter.api.Assertions.\*;  
    import static org.mockito.Mockito.\*;  
      
    class BuildControllerTest {  
     private BuildController buildController;  
     private BuildService buildService;  
      
     @BeforeEach  
     void setUp(){  
     buildService = *mock*(BuildServiceImpl.class);  
     buildController = new BuildController(buildService);  
     }  
      
      
     private Imageadditionaldocument getImageAdditionalDocument(){  
     return new Imageadditionaldocument(  
     1L,  
     "jhvjhvjh",  
     Date.*from*(Instant.*now*()),  
     new Agent(),  
     new Client(),  
     new Cni()  
     );  
     }  
     private Cni getReferenceDocument() {  
     return new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     }  
      
     private Page<Referencedocument> listDocReadyToBuildData(){  
     Cni cni = getReferenceDocument();  
     Cni cni1 = new Cni(  
     89L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     Cni cni2 = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     198L,  
     "kjjlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     Cni cni3 = new Cni(  
     889L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     7L,  
     "jlkfsf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Ready*,  
     null  
     );  
     List<Referencedocument> referencedocumentList = new ArrayList<>();  
     referencedocumentList.add(cni);  
     referencedocumentList.add(cni1);  
     referencedocumentList.add(cni2);  
     referencedocumentList.add(cni3);  
      
     return new PageImpl<Referencedocument>(referencedocumentList, PageRequest.*of*(0, 2),4L);  
      
     }  
      
     private Page<Referencedocument> listDocInRealTimeBuildData(){  
     Cni cni = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni1 = new Cni(  
     89L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     1L,  
     "jlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni2 = new Cni(  
     1L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     198L,  
     "kjjlkfsdf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     Cni cni3 = new Cni(  
     889L,  
     null,  
     "123456",  
     "lastName",  
     "sdvsdv",  
     Date.*from*(Instant.*now*()),  
     Gender.*MALE*,  
     "profession",  
     "nameOfFather",  
     "nameOfMother",  
     Date.*from*(Instant.*now*()),  
     Date.*from*(Instant.*now*()),  
     "address",  
     new Agent(  
     2L,  
     "jkhkdfjhk@sd.sd",  
     PersonStatus.*ACTIVE*,  
     "jhbfbsdjfb",  
     "bjbnbvwsds",  
     "login",  
     "pwd"  
     ),  
     new Client(  
     7L,  
     "jlkfsf@gm.de",  
     PersonStatus.*ACTIVE*,  
     "123456",  
     Date.*from*(Instant.*now*())  
     ),  
     this.getImageAdditionalDocument(),  
     StatusTreatmentSystemeList.*BUILD*,  
     StatusInTreatment.*Waiting*,  
     null  
     );  
     List<Referencedocument> referencedocumentList = new ArrayList<>();  
     referencedocumentList.add(cni);  
     referencedocumentList.add(cni1);  
     referencedocumentList.add(cni2);  
     referencedocumentList.add(cni3);  
      
     return new PageImpl<Referencedocument>(referencedocumentList, PageRequest.*of*(0, 2),4L);  
      
     }  
      
      
     @Test  
     void givenPageAndSize\_itShouldReturnAllDocumentReadyToPrint() {  
     //Given  
     int page= 0;  
     int size = 2;  
      
     //When  
     *when*(buildService.listDocReadyToBuild(page,size)).thenReturn(this.listDocReadyToBuildData());  
     ResponseEntity<?> responseEntity = buildController.findAllDocumentReadyToPrint(page,size);  
      
     //Then  
     *assertThat*(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     assertThat(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.OK);  
     assertThat(responseEntity.getBody().toString()).isEqualTo(this.listDocReadyToBuildData().toString());  
     }  
      
     @Test  
     void givenPageAndSize\_itShouldThrowAnExceptionDocumentReadyToPrint() {  
     //Given  
     int page= 0;  
     int size = 2;  
      
     //When  
     when(buildService.listDocReadyToBuild(page,size)).thenThrow(new MockitoException("Error while getting data"));  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->buildController.findAllDocumentReadyToPrint(page,size)  
     );  
     }  
      
     @Test  
     void givenIdAndStatusInTreatment\_itShouldUpdateDocumentStatus() {  
     //Given  
     Long id = 1L;  
     StatusInTreatment statusInTreatment = StatusInTreatment.Ready;  
      
     //When  
     doNothing().when(mock(BuildServiceImpl.class)).changeStatusOfDocument(id, statusInTreatment);  
     ResponseEntity<?> responseEntity = buildController.updateDocumentStatus(id, statusInTreatment);  
     assertThat(responseEntity.getStatusCodeValue()).isEqualTo(204);  
     assertThat(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.NO\_CONTENT);  
     }  
      
     @Test  
     void givenPageAndSize\_itShouldReturnAllDocumentRealTimePrinting() {  
     //Given  
     int page= 0;  
     int size = 2;  
      
     //When  
     when(buildService.listDocInRealTimeBuilding(page,size)).thenReturn(this.listDocReadyToBuildData());  
     ResponseEntity<?> responseEntity = buildController.findAllDocumentRealTimePrinting(page,size);  
      
     //Then  
     assertThat(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     assertThat(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.OK);  
     assertThat(responseEntity.getBody().toString()).isEqualTo(this.listDocReadyToBuildData().toString());  
     }  
      
     @Test  
     void givenPageAndSize\_itShouldThrowAnExceptionDocumentRealTimePrinting() {  
     //Given  
     int page= 0;  
     int size = 2;  
      
     //When  
     when(buildService.listDocInRealTimeBuilding(page,size)).thenThrow(new MockitoException("Error while getting data"));  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->buildController.findAllDocumentRealTimePrinting(page,size)  
     );  
     }  
      
     @Test  
     void givenId\_itShouldThrowAnExceptionWhenFindDocumentById() {  
     //Given  
     Long id= 1L;  
      
     //When  
     when(buildService.getDocumentInformationById(id)).thenThrow(new MockitoException("Error while getting data"));  
     Throwable throwable = assertThrows(  
     Exception.class,  
     ()->buildController.findDocumentById(id)  
     );  
     }  
      
     @Test  
     void givenId\_itShouldReturnDocument() {  
     //Given  
     Long id= 1L;  
      
     //When  
     when(buildService.getDocumentInformationById(id)).thenReturn(this.getReferenceDocument());  
     ResponseEntity<?> responseEntity = buildController.findDocumentById(id);  
      
     //Then  
     assertThat(responseEntity.getStatusCodeValue()).isEqualTo(200);  
     assertThat(responseEntity.getStatusCode()).isEqualByComparingTo(HttpStatus.OK);  
     assertThat(responseEntity.getBody().toString()).isEqualTo(this.getReferenceDocument().toString());  
     }  
    }