Проект: PrintMatic

Тестове

**AdminServiceTest**

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| **1.1** | **grantRole\_WhenUserNotFound\_ReturnsNotFoundMessage** | - Система с празна или неподходяща база данни <br/> - Имейл, несъществуващ като потребител | 1. Извикване на adminService.grantRole("test@example.com", RoleEnum.EMPLOYEE) <br/> 2. userRepository.findByEmail(...) връща Optional.empty() | Методът връща MessageResponseDTO със статус **404** и съобщение **"User whit this email not found"**, без да се извиква save(...) в userRepository. |
| **1.2** | **grantRole\_WhenRoleAlreadyGranted\_ReturnsBadRequestMessage** | - Съществуващ потребител с вече зададена роля | 1. Зареждане на потребител с роля EMPLOYEE <br/> 2. Извикване на adminService.grantRole(...) | Методът връща статус **400** и съобщение **"This role already is granted to this user"**, без да се правят промени в базата. |
| **1.3** | **grantRole\_Success** | - Съществуващ потребител без дадената роля | 1. Намиране на потребител по имейл <br/> 2. Намиране на роля по име <br/> 3. Добавяне на ролята към потребителя <br/> 4. Запис в базата | Методът връща статус **200** и съобщение **"Role successfully granted to the user"**. В базата ролята е добавена на потребителя. |
| **1.4** | **searchUser\_ReturnsMatchingUsers** | - Налични няколко потребители в базата | 1. Извикване на adminService.searchUser("john") <br/> 2. userRepository.findAll() <br/> 3. Филтриране/мапване към DTO | Връща се списък от DTO, поне един от които съдържа "john" в имейла. |
| **1.5** | **removeRole\_WhenUserNotFound\_ReturnsNotFoundMessage** | - Имейл, който не съществува в базата данни | 1. Извикване на adminService.removeRole("test@example.com", RoleEnum.EMPLOYEE) <br/> 2. userRepository.findByEmail(...) връща Optional.empty() | Методът връща статус **404** и съобщение **"User not found"**, не се извиква save(...). |
| **1.6** | **removeRole\_WhenRoleNotGranted\_ReturnsNotFoundMessage** | - Съществуващ потребител, но без съответната роля | 1. Зареждане на потребител <br/> 2. Зареждане на роля <br/> 3. Ролята я няма прикачена към потребителя <br/> 4. removeRole(...) | Методът връща статус **404** и съобщение **"This role is not granted to this user"**, не се извършва запис. |
| **1.7** | **removeRole\_Success** | - Съществуващ потребител с вече дадена роля | 1. Намиране на потребител по имейл <br/> 2. Ролята е прикачена <br/> 3. removeRole(...) <br/> 4. Запис в базата | Методът връща статус **200** и съобщение **"Role successfully removed from this user"** и реално премахва ролята и от потребителя, и от списъка с потребители в ролята. |

**DocumentAnalysisServiceTest**

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| **2.1** | **analyzeDocument\_ColorPDF\_Success** | - Създадено цветно PDF-файлче като MockMultipartFile | 1. Изпращаме цветно PDF към analyzeDocument(file, false) <br/> 2. Извършва се анализ на съдържанието | FileAnalysisResultDTO с totalPages = 1, colorfulPages = 1, grayscalePages = 0. |
| **2.2** | **analyzeDocument\_GrayscalePDF\_Success** | - Създадено черно-бяло PDF-файлче | 1. Изпращаме grayscale PDF към analyzeDocument(file, false) | FileAnalysisResultDTO с totalPages = 1, colorfulPages = 0, grayscalePages = 1. |
| **2.3** | **analyzeDocument\_ForcedGrayscalePDF\_Success** | - Цветно PDF-файлче, но параметър forcedGrayscale = true | 1. Подаваме цветен PDF <br/> 2. Задаваме принудително grayscale | FileAnalysisResultDTO показва colorfulPages = 0, grayscalePages = 1. |
| **2.4** | **analyzeDocument\_ColorImage\_Success** | - Цветно JPEG изображение | 1. Подаваме цветно изображение <br/> 2. Методът разпознава, че е цветно | FileAnalysisResultDTO с totalPages = 1, colorfulPages = 1, grayscalePages = 0. |
| **2.5** | **analyzeDocument\_GrayscaleImage\_Success** | - Черно-бяло изображение | 1. Подаваме grayscale изображение <br/> 2. Методът разпознава, че е черно-бяло | FileAnalysisResultDTO с totalPages = 1, colorfulPages = 0, grayscalePages = 1. |
| **2.6** | **analyzeDocument\_UnsupportedFile\_ThrowsException** | - Файл с неподдържано разширение (напр. .txt) | 1. Подаваме MockMultipartFile с text/plain <br/> 2. Методът проверява типа | Хвърля се IOException (или грешка) при неподдържан формат. |
| **2.7** | **isPdf\_ValidPDF\_ReturnsTrue** | - Име на файл с разширение .pdf | 1. Изпращаме isPdf("test.pdf") | Методът връща true. |
| **2.8** | **isPdf\_InvalidPDF\_ReturnsFalse** | - Име на файл без .pdf | 1. Изпращаме isPdf("test.jpg") | Методът връща false. |

**MailServiceTest**

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| **3.1** | **sendEmail\_Success** | - Конфигуриран JavaMailSender, MimeMessage | 1. Подготвяме валиден MimeMessage <br/> 2. Извикваме mailService.sendEmail(...) | Не хвърля изключение; mailSender.send(...) се извиква успешно. |
| **3.2** | **sendEmail\_ThrowsException** | - Принудително хвърляне на грешка при mailSender.send(...) | 1. mailSender.send(...) хвърля RuntimeException <br/> 2. Извикваме mailService.sendEmail(...) | Хвърля се RuntimeException с очаквано съобщение, методът не успява да прати имейл. |
| **3.3** | **createHtmlBody\_Success** | - Създаден OrderEntity и UserEntity с нормални стойности | 1. Извикваме mailService.createHtmlBody(order, user) <br/> 2. Проверяваме съдържанието | Връща се HTML низ с коректни данни за поръчката (име, цена и т.н.). |
| **3.4** | **createHtmlBody\_WithSingleSided** | - OrderEntity с doubleSided=false | 1. Извикваме createHtmlBody(...) <br/> 2. Проверяваме наличието на "No" в HTML-a | HTML резултатът показва Single-sided (No). |
| **3.5** | **createHtmlBody\_ContainsAllRequiredSections** | - OrderEntity с разнородни данни (A3, doubleSided и др.) | 1. Извикваме createHtmlBody(...) <br/> 2. Проверяваме дали в HTML има всички части: header, content, order-details, footer и т.н. | HTML резултатът включва всички очаквани под-секции и данни. |

**OrderServiceTest**

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| **4.1** | **createOrder\_Success** | - Съществуващ потребител, достатъчна информация в DTO | 1. Извикваме orderService.createOrder(dto, principal) <br/> 2. Параметрите (pages, copies) са валидни | Резултат OrderResultDTO със статус **200**, връщащ генерирана цена. |
| **4.2** | **createOrder\_UserNotFound** | - Principal с несъществуващ имейл | 1. Извикваме orderService.createOrder(...) <br/> 2. userRepository.findByEmail(...) връща празно | Резултат OrderResultDTO със статус **404** и съобщение "User not found". |
| **4.3** | **createOrder\_InvalidPageCount** | - Sum(colorPages + grayscalePages) ≠ totalPages | 1. Подаваме DTO, където color + grayscale != total <br/> 2. Извикваме createOrder(...) | Връща се OrderResultDTO със статус **400** и съобщение за невалидни страници. |
| **4.4** | **updateOrderStatus\_Success** | - Съществуваща поръчка в статус, позволяващ промяна | 1. Извикваме updateOrderStatus(1L, IN\_PROGRESS) <br/> 2. Запис в базата | Връща се MessageResponseDTO със статус **200**, статусът на поръчката е обновен. |
| **4.5** | **updateOrderStatus\_OrderNotFound** | - Невалидно orderId | 1. orderRepository.findById(...) връща празно <br/> 2. Извикваме updateOrderStatus(...) | Връща се MessageResponseDTO със статус **404** и не се извършва запис. |
| **4.6** | **updateOrderStatus\_InvalidStatusTransition** | - Поръчка в COMPLETED, опит за връщане в IN\_PROGRESS | 1. Намираме поръчка със статус COMPLETED <br/> 2. Извикваме updateOrderStatus(...) | Връща се статус **400**, защото преходът е невалиден. |
| **4.7** | **updateOrderStatus\_RefundOnCancellation** | - Поръчка в PENDING със зададена цена | 1. Извикваме updateOrderStatus(..., CANCELED) <br/> 2. Проверяваме дали балансът на потребителя се възстановява | След извикването статусът е CANCELED, потребителят получава обратно сумата. |
| **4.8** | **getOrdersOfUser\_Success** | - Има поне една поръчка на потребителя | 1. Извикваме getOrdersOfUser(principal, pageRequest) <br/> 2. Проверяваме върнатия Page<UserOrderDTO> | Връща се непразен списък от поръчки на съответния user, с коректно генерирани download URL. |
| **4.9** | **getOrders\_Success** | - Наличие на поръчки със статуси PENDING или IN\_PROGRESS | 1. Извикваме getOrders(SortBy.DEADLINE, pageRequest) <br/> 2. Проверка на съдържанието | Връща се страница с OrderDTO за поръчките с очакваните статуси. |
| **4.10** | **getOrderById\_Success** | - Съществуваща поръчка с ID=1 | 1. orderRepository.findById(1L) връща OrderEntity <br/> 2. Преобразуваме към DTO | Optional<OrderDTO> е isPresent()==true и съдържа поръчката. |
| **4.11** | **getOrderById\_NotFound** | - Невалидно ID | 1. orderRepository.findById(...) връща Optional.empty() <br/> 2. Връщаме Optional.empty() | Резултатът е празен Optional, без грешка. |

**PaymentServiceIntegrationTest**

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| **5.1** | **createSessionForPayingOrder\_ShouldCreateValidSession** | - Налична поръчка със статус UNPAID | 1. Извикваме paymentService.createSessionForPayingOrder(orderId, principal) <br/> 2. Връща SessionResponseDTO | Създава валидна Stripe сесия, има sessionId и stripePaymentURL. |
| **5.2** | **payFromBalance\_ShouldSucceedWithSufficientBalance** | - User има достатъчно средства | 1. Извикваме payFromBalance(orderId, principal) <br/> 2. Проверяваме дали стига балансът | Поръчката става PENDING, балансът на user намалява, методът връща статус **200**. |
| **5.3** | **payFromBalance\_ShouldFailWithInsufficientBalance** | - User има баланс по-малък от цената на поръчката | 1. Извикваме payFromBalance(...) <br/> 2. Проверяваме дали хвърля грешка/връща 500 | Статус **500**, съобщение "Payment not successful", поръчката остава UNPAID. |

**UserServiceTest**

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| 6.1 | registerUser\_Success | Базата данни не съдържа потребител с дадения email. | 1. Извикаваме метода registerUser с валиден RegistrationDTO. 2. Уверете се, че userRepository.save се извиква. 3. Проверяваме, че връщаният резултат е Optional с данни. | Успешна регистрация и записване на потребителя в базата данни. |
| 6.2 | registerUser\_EmailExists | Съществува потребител с дадения email в базата данни. | 1. Извикваме метода registerUser с RegistrationDTO, съдържащ email, който вече съществува. 2. Проверяваме, че userRepository.save не се извиква. | Връщане на празен Optional, без промени в базата данни. |
| 6.3 | getCurrentUser\_Success | Валиден principal, който съответства на съществуващ потребител. | 1. Извикваме метода getCurrentUser с валиден Principal. 2. Проверяваме, че потребителят е намерен и мапнат към UserDTO. | Успешно връщане на UserDTO за текущия потребител. |
| 6.4 | updateCurrentUser\_Success | Съществуващ потребител в базата данни, principal съвпадащ с email на потребителя, и новият email не съществува. | 1. Извикваме метода updateCurrentUser с валиден UpdateUserDTO и Principal. 2. Уверяваме се, че userRepository.save се извиква. 3. Проверяваме, че статуса и съобщението в резултата са коректни. | Статус 200 и съобщение "User updated successfully". |
| 6.5 | updateCurrentUser\_UserNotFound | Principal не съвпада с никой потребител в базата данни. | 1. Извикваме метода updateCurrentUser с валиден UpdateUserDTO, но невалиден Principal. 2. Проверяваме, че userRepository.save не се извиква. 3. Уверяваме се, че статусът и съобщението отразяват "User not found". | Статус 404 и съобщение "User not found". |
| 6.6 | updateCurrentUser\_EmailAlreadyExists | Новият email вече съществува в базата данни. | 1. Извикваме метода updateCurrentUser с UpdateUserDTO, съдържащ email, който вече съществува. 2. Проверяваме, че userRepository.save не се извиква. 3. Уверяваме, че статусът и съобщението отразяват "User with this email already exists". | Статус 400 и съобщение "User with this email already exists". |
| 6.7 | seedUsers\_Success | Базата данни е празна (няма записани потребители). | 1. Извикваме метода seedUsers. 2. Уверяваме, че roleRepository.save е извикан три пъти. 3. Уверяваме, че userRepository.save е извикан. | Успешно създаване на роли и потребители в базата данни. |
| 6.8 | seedUsers\_SkipWhenUsersExist | В базата данни има вече съществуващи потребители. | 1. Извикваме метода seedUsers. 2. Проверяваме, че roleRepository.save и userRepository.save не се извикват. | Методът пропуска създаване на роли и потребители, когато вече съществуват. |

UserControllerTest

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| 7.1 | Успешна регистрация | Празна база данни (премахнати потребители). | 1. Извикваме API /api/user/register с валиден RegistrationDTO. 2. Проверяваме, че API връща статус 200. 3. Проверяваме, че потребителят е добавен в базата данни. 4. Проверяваме хеширането на паролата. | Статус 200 и добавен потребител с валидни данни в базата. |
| 7.2 | Неуспешна регистрация | Вече съществуващ потребител или невалиден email. | 1. Извикваме API /api/user/register с невалиден RegistrationDTO. 2. Провери, че API връща статус 400. | Статус 400 (Bad Request). |
| 7.3 | Успешно влизане в системата | Създаден потребител с email: john@doe.com и парола: password. | 1. Извикайте API /api/user/login с валиден LoginDTO. 2. Провери, че API връща статус 200. | Статус 200 и успешен вход в системата. |
| 7.4 | Неуспешно влизане в системата | Създаден потребител. | 1. Извикваме API /api/user/login с грешна комбинация от email и парола. 2. Проверяваме, че API връща статус 403. 3. Тестваме с празни или невалидни стойности за email/парола. 4. Проверяваме статус за всяка ситуация. | Статус 403 (Forbidden) при всяко неуспешно влизане. |

AdminControllerTest

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| 8.1 | testGrantRole\_Success | Съществува потребител без ролята EMPLOYEE. Администраторът е логнат с валиден токен. | 1. Създаваме потребител без ролята EMPLOYEE. 2. Влизаме като администратор и взимаме токена. 3. Извикваме /api/admin/grantRole с валидния токен. 4. Проверяваме, че ролята е добавена успешно. | Ролята EMPLOYEE е успешно добавена към потребителя. |
| 8.2 | testGrantRole\_Unauthorized\_NoToken | Не е предоставен токен за достъп. | 1. Опитваме се да извикате /api/admin/grantRole без токен. 2. Проверяваме, че статусът на отговора е 403 Forbidden. | Достъпът е отказан с статус 403 Forbidden. |
| 8.3 | testGrantRole\_Unauthorized\_NonAdminToken | Обикновен потребител е логнат, но не е администратор. | 1. Влизаме като обикновен потребител. 2. Извикваме /api/admin/grantRole с токена на обикновения потребител. 3. Проверяваме, че статусът на отговора е 403 Forbidden. | Достъпът е отказан с статус 403 Forbidden. |
| 8.4 | testRemoveRole\_Success | Съществува потребител с ролята EMPLOYEE. Администраторът е логнат с валиден токен. | 1. Създава се потребител с ролята EMPLOYEE. 2. Влизане като администратор и вземете токена. 3. Извиква се /api/admin/removeRole с валидния токен. 4. Проверява се че ролята е премахната успешно. | Ролята EMPLOYEE е успешно премахната от потребителя. |
| 8.5 | testRemoveRole\_Unauthorized\_EmployeeToken | Потребител с ролята EMPLOYEE се опитва да премахне роля. | 1. Логване като потребител с ролята EMPLOYEE. 2. Извикване на /api/admin/removeRole с токена на служителя. 3. Проверява се, че статусът на отговора е 403 Forbidden и че ролята не е премахната. | Достъпът е отказан с статус 403 Forbidden. |
| 8.6 | testRemoveRole\_Unauthorized\_NormalUserToken | Обикновен потребител (без роли) се опитва да премахне роля. | 1. Логва се като обикновен потребител. 2. Извиква се /api/admin/removeRole с токена на обикновения потребител. 3. Проверява се, че статусът на отговора е 403 Forbidden и че ролята не е премахната. | Достъпът е отказан с статус 403 Forbidden. |
| 8.7 | testSearchUser\_Success | Съществува потребител, който съответства на зададения критерий за търсене. Администраторът е логнат с валиден токен. | 1. Създайте потребител с email, който съответства на търсената стойност. 2. Влезте като администратор и вземете токена. 3. Извикайте /api/admin/search/email с валидния токен. 4. Проверете, че резултатът съдържа правилния потребител. | Успешно връщане на потребителите, съответстващи на критерия за търсене. |

OrderControllerTest

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| 9.1 | createOrder\_Success | Съществува потребител с валиден токен и валиден OrderCreationDTO. | 1. Влезте в системата и вземете токен за потребителя. 2. Извикайте /api/order/create с валиден OrderCreationDTO и токен. 3. Проверете, че статусът е 200 OK. 4. Проверете, че резултатът съдържа orderId и цена. | Успешно създаване на поръчка със статус 200 OK и връщане на данни за поръчката. |
| 9.2 | createOrder\_InvalidData | Съществува потребител с валиден токен, но невалиден OrderCreationDTO. | 1. Променете данните в OrderCreationDTO на невалидни (например, отрицателен брой копия). 2. Извикайте /api/order/create с невалидния DTO. 3. Проверете, че статусът е 400 Bad Request. | Връща се статус 400 Bad Request. |
| 9.3 | getOrder\_Success | Съществува създадена поръчка и потребител с валиден токен. | 1. Създайте поръчка с /api/order/create. 2. Вземете orderId от създадената поръчка. 3. Извикайте /api/order/{orderId} с валиден токен. 4. Проверете, че статусът е 200 OK и данните съвпадат. | Успешно връщане на поръчката със статус 200 OK. |
| 9.4 | updateOrderStatus\_Success | Съществува администратор с валиден токен и създадена поръчка. | 1. Създайте поръчка с /api/order/create. 2. Влезте като администратор и вземете токен. 3. Извикайте /api/order/updateOrderStatus/{orderId} с токен и валиден статус. 4. Проверете, че статусът е актуализиран успешно. | Успешна актуализация на статуса със статус 200 OK. |
| 9.5 | updateOrderStatus\_Unauthorized | Обикновен потребител се опитва да актуализира статус на поръчка. | 1. Създайте поръчка с /api/order/create. 2. Влезте като обикновен потребител. 3. Извикайте /api/order/updateOrderStatus/{orderId} с токен на обикновения потребител. 4. Проверете, че статусът е 403 Forbidden. | Достъпът е отказан със статус 403 Forbidden. |
| 9.6 | getUserOrders\_Success | Съществуват създадени поръчки за потребител с валиден токен. | 1. Създайте поръчка с /api/order/create. 2. Извикайте /api/order/user с валиден токен. 3. Проверете, че статусът е 200 OK и поръчките са върнати като масив. | Успешно връщане на поръчките на потребителя със статус 200 OK. |
| 9.7 | getPendingOrInProgress\_Success | Съществуват поръчки със статус PENDING или IN\_PROGRESS, и администраторът има валиден токен. | 1. Създайте поръчка с /api/order/create. 2. Влезте като администратор и вземете токен. 3. Извикайте /api/order/getPendingOrInProgress с валиден токен. 4. Проверете, че статусът е 200 OK и резултатите съдържат очакваните поръчки. | Успешно връщане на поръчките със статус 200 OK. |
| 9.8 | getPendingOrInProgress\_Unauthorized | Обикновен потребител се опитва да достъпи поръчки със статус PENDING или IN\_PROGRESS. | 1. Влезте като обикновен потребител и вземете токен. 2. Извикайте /api/order/getPendingOrInProgress с токена на потребителя. 3. Проверете, че статусът е 403 Forbidden. | Достъпът е отказан със статус 403 Forbidden. |

PaymentControllerTest

| **Номер** | **Описание** | **Предварителни изисквания за да бъде проведен теста** | **Стъпки за изпълнение** | **Очакван резултат** |
| --- | --- | --- | --- | --- |
| 10.1 | testCreateOrderSession\_Success | Съществува създадена поръчка със статус UNPAID и потребител с валиден токен. | 1. Извикайте /api/payment/createOrderSession/{orderId} с валиден токен на потребителя. 2. Проверете, че статусът е 200 OK и са върнати sessionId и stripePaymentURL. | Успешно създаване на Stripe сесия със статус 200 OK. |
| 10.2 | testCreateOrderSession\_OrderNotFound | Поръчката с даденото orderId не съществува. | 1. Извикайте /api/payment/createOrderSession/9999 с валиден токен. 2. Проверете, че статусът е 500 Internal Server Error. | Статус 500 Internal Server Error. |
| 10.3 | testOrderSuccess\_ChargeSucceeded\_WithFakeSession | Съществува създадена поръчка и потребител с валиден токен. | 1. Създайте OrderPaymentSuccessDTO с валидно orderId и fake\_stripe\_session\_id. 2. Извикайте /api/payment/orderSuccess с валиден токен. 3. Проверете, че статусът е 200 OK или 500 Internal Server Error. | Статус 200 OK или 500 Internal Server Error. |
| 10.4 | testAddToBalanceSession\_Success | Съществува потребител с валиден токен. | 1. Извикайте /api/payment/addToBalanceSession с параметър amount=10.00 и валиден токен. 2. Проверете, че статусът е 200 OK и са върнати sessionId и stripePaymentURL. | Успешно създаване на Stripe сесия за баланс със статус 200 OK. |
| 10.5 | testDepositBalanceSuccess\_FakeStripeId | Съществува потребител с валиден токен и невалиден Stripe ID. | 1. Извикайте /api/payment/depositBalanceSuccess с stripeId=fake\_stripe\_id и валиден токен. 2. Проверете, че статусът е 500 Internal Server Error. | Статус 500 Internal Server Error. |
| 10.6 | testPayOrderFromBalance\_NotEnoughBalance | Съществува потребител с недостатъчен баланс и валиден токен. | 1. Извикайте /api/payment/payOrderFromBalance/{orderId} с валиден токен. 2. Проверете, че статусът е 500 Internal Server Error и съобщението съдържа информация за недостатъчен баланс. | Статус 500 Internal Server Error със съобщение за неуспешно плащане. |
| 10.7 | testGetPayments\_Empty | Съществува потребител с валиден токен и без създадени плащания. | 1. Извикайте /api/payment/getPayments с валиден токен. 2. Проверете, че статусът е 200 OK и върнатото съдържание е празен масив. | Статус 200 OK и празен масив в съдържанието. |
| 10.8 | testCreateOrderSession\_RealStripe | Съществува създадена поръчка със статус UNPAID и потребител с валиден токен за реална Stripe сесия. | 1. Извикайте /api/payment/createOrderSession/{orderId} с валиден токен. 2. Проверете, че статусът е 200 OK и са върнати реални sessionId и stripePaymentURL. 3. Съхранете sessionId за последващи тестове. | Успешно създаване на реална Stripe сесия със статус 200 OK. |
| 10.9 | testOrderSuccess\_AfterManualTestPayment | Реална Stripe сесия е създадена и е извършено ръчно плащане. | 1. Създайте OrderPaymentSuccessDTO с валидно orderId и sessionId от предишния тест. 2. Извикайте /api/payment/orderSuccess с валиден токен. 3. Проверете, че статусът е 200 OK или 500 Internal Server Error. | Статус 200 OK или 500 Internal Server Error. |

| **Номер на тестов случай** |  | **Версия на програмата** | **Успешно преминаване на теста** | **Получен резултат, различаващ се от очаквания** | **Име на студента, извършил теста** |
| --- | --- | --- | --- | --- | --- |
| 1.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 1.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 1.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 1.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 1.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 1.6 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 1.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.6 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 2.8 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 3.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 3.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 3.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 3.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 3.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.6 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.8 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.9 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.10 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 4.11 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 5.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 5.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 5.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.6 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 6.8 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 7.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 7.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 7.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 7.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.5 |  | 1.0 | Не | java.lang.AssertionError: Status expected:<403> but was:<200> | Симеон Кюрчийски |
| 8.6 |  | 1.0 | Не | java.lang.AssertionError: Status expected:<403> but was:<200> | Симеон Кюрчийски |
| 8.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.8 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 8.9 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.6 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 9.8 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.1 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.2 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.3 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.4 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.5 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.6 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.7 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.8 |  | 1.0 | Да | Не | Симеон Кюрчийски |
| 10.9 |  | 1.0 | Да | Не | Симеон Кюрчийски |