Отчет об юнит-тестировании

1. Информация о тестировании

Название проекта: travel-planner-backend

Версия API: v1

Дата тестирования: 05.05.2025

1. Описание тестирования

Проведено юнит-тестирование с использованием JUnit. Тестировалась функциональность обработчиков API у сервисов: auth, external, library, planner.

1. Результаты тестирования
   1. Сервис auth
      1. Успешная регистрация пользователя (registerUser\_whenSuccess\_shouldSaveUserAndSendVerificationEmail)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешная регистрация пользователя |
| **Цель** | Проверить, что при успешной регистрации пользователя: данные корректно сохраняются в репозиторий, генерируется и отправляется токен подтверждения email. |
| **Функция** | AuthServiceImpl.registerUser |
| **Предусловия** | Моки UserRepository (проверка email/username на уникальность возвращает false, save возвращает пользователя с ID), PasswordEncoder (возвращает закодированный пароль), EmailService (для отправки письма). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | RegisterRequest (с email, username, password), ipAddress, deviceInfo. |
| **Шаги выполнения** | 1. Подготовить RegisterRequest и другие входные данные.   2. Настроить моки (см. Предусловия).   3. Вызвать authService.registerUser(request, ipAddress, deviceInfo).   4. Перехватить результат (ID пользователя) и захватить аргументы моков (userCaptor, stringCaptor). |
| **Ожидаемый результат** | Метод возвращает строковый ID пользователя. Пользователь сохраняется с корректными данными (email, username, зашифрованный пароль, isVerified=false, isAdmin=false, временные метки). Вызывается emailService.sendVerificationEmail с email, username и сгенерированным токеном (UUID). |
| **Проверка** | Сравнить возвращенный ID с ожидаемым. С помощью userCaptor проверить поля сохраненного пользователя. С помощью stringCaptor проверить, что токен для sendVerificationEmail не null и является валидным UUID. Проверить вызовы userRepository.existsByEmail, userRepository.existsByUsername, passwordEncoder.encode, userRepository.save, emailService.sendVerificationEmail. Убедиться, что jwtTokenProvider не вызывался. |
| **Полученный результат** | Функция успешно вернула ID пользователя. Пользователь сохранен корректно. Email-подтверждение отправлено с валидным токеном. Вызовы зависимостей корректны. |

* + 1. Регистрация с существующим email (registerUser\_whenEmailExists\_shouldThrowConflictException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Регистрация с существующим email |
| **Цель** | Проверить, что при попытке регистрации с уже существующим email выбрасывается исключение ResponseStatusException со статусом 409 (Conflict). |
| **Функция** | AuthServiceImpl.registerUser |
| **Предусловия** | Мок UserRepository. userRepository.existsByEmail возвращает true. |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | RegisterRequest (с существующим email), ipAddress, deviceInfo. |
| **Шаги выполнения** | 1. Подготовить RegisterRequest.   2. Настроить userRepository.existsByEmail на возврат true.   3. Вызвать authService.registerUser внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 409 и сообщением, содержащим "email". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Проверить, что методы existsByUsername, encode, save, sendVerificationEmail не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (409 Conflict). Лишние вызовы зависимостей отсутствуют. |

* + 1. Регистрация с существующим username (registerUser\_whenUsernameExists\_shouldThrowConflictException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Регистрация с существующим username |
| **Цель** | Проверить, что при попытке регистрации с уже существующим username выбрасывается исключение ResponseStatusException со статусом 409 (Conflict). |
| **Функция** | AuthServiceImpl.registerUser |
| **Предусловия** | Мок UserRepository. userRepository.existsByEmail возвращает false, userRepository.existsByUsername возвращает true. |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | RegisterRequest (с существующим username), ipAddress, deviceInfo. |
| **Шаги выполнения** | 1. Подготовить RegisterRequest.   2. Настроить userRepository.existsByEmail(false) и userRepository.existsByUsername(true).   3. Вызвать authService.registerUser внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 409 и сообщением, содержащим "username". |
| **Проверка** | Проверить тип и статус исключения. Проверить, что вызывался existsByEmail, но не вызывались encode, save, sendVerificationEmail. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (409 Conflict). Лишние вызовы зависимостей отсутствуют. |

* + 1. Успешное подтверждение email (verifyEmail\_whenTokenIsValid\_shouldVerifyUserAndReturnAuthResponse)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное подтверждение email |
| **Цель** | Проверить, что при валидном токене подтверждения: пользователь помечается как верифицированный, генерируется AuthResponse с токенами и информацией о пользователе. |
| **Функция** | AuthServiceImpl.verifyEmail |
| **Предусловия** | Моки EmailService (возвращает неверифицированного пользователя по токену), UserRepository (для сохранения обновленного пользователя), JwtTokenProvider (генерация токенов), UserService (маппинг в DTO). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | validToken (строка), ipAddress, deviceInfo. |
| **Шаги выполнения** | 1. Настроить моки. emailService.verifyEmailToken возвращает userToVerify.   2. Вызвать authService.verifyEmail(validToken, ipAddress, deviceInfo).   3. Перехватить результат (AuthResponse) и захватить аргумент userRepository.save. |
| **Ожидаемый результат** | Метод возвращает AuthResponse с accessToken, refreshToken, expiresIn и UserInfoDto. Пользователь сохраняется с isVerified=true. |
| **Проверка** | Проверить поля AuthResponse (токены, user DTO с emailVerified=true). С помощью userCaptor проверить, что у сохраненного пользователя isVerified=true и обновилась метка updatedAt. Проверить вызовы jwtTokenProvider.generateAccessToken, jwtTokenProvider.generateRefreshToken, userService.mapToUserInfoDto. |
| **Полученный результат** | Функция успешно вернула AuthResponse. Пользователь помечен как верифицированный. Вызовы зависимостей корректны. |

* + 1. Подтверждение email с невалидным токеном (verifyEmail\_whenTokenIsInvalid\_shouldThrowException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Подтверждение email с невалидным токеном |
| **Цель** | Проверить, что при невалидном токене подтверждения выбрасывается исключение ResponseStatusException (400 Bad Request). |
| **Функция** | AuthServiceImpl.verifyEmail |
| **Предусловия** | Мок EmailService. emailService.verifyEmailToken настроен на выброс ResponseStatusException. |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | invalidToken (строка), ipAddress, deviceInfo. |
| **Шаги выполнения** | 1. Настроить emailService.verifyEmailToken на выброс исключения.   2. Вызвать authService.verifyEmail внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 400 и сообщением "Неверный или истекший токен". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Убедиться, что userRepository.save, jwtTokenProvider.generate..., userService.mapToUserInfoDto не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (400 Bad Request). Лишние вызовы зависимостей отсутствуют. |

* + 1. Создание ответа аутентификации (успешный логин) (createAuthResponse\_whenUserExists\_shouldReturnAuthResponse)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Создание ответа аутентификации (успешный логин) |
| **Цель** | Проверить, что при наличии пользователя с заданным email: обновляется время последнего входа, генерируется AuthResponse с токенами и информацией о пользователе. |
| **Функция** | AuthServiceImpl.createAuthResponse |
| **Предусловия** | Моки UserService (поиск пользователя возвращает Optional.of(foundUser), updateLastLogin возвращает пользователя с обновленной меткой, mapToUserInfoDto возвращает DTO), JwtTokenProvider (генерация токенов). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | email (строка), ipAddress, deviceInfo, deviceId. |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.createAuthResponse(email, ipAddress, deviceInfo, deviceId).   3. Перехватить результат (AuthResponse). |
| **Ожидаемый результат** | Метод возвращает AuthResponse с accessToken, refreshToken, expiresIn=3600 и UserInfoDto. Вызывается userService.updateLastLogin. |
| **Проверка** | Проверить поля AuthResponse (токены, DTO пользователя). Проверить вызовы userService.findByEmail, userService.updateLastLogin, jwtTokenProvider.generateAccessToken, jwtTokenProvider.generateRefreshToken, userService.mapToUserInfoDto. |
| **Полученный результат** | Функция успешно вернула AuthResponse. Время последнего входа обновлено. Вызовы зависимостей корректны. |

* + 1. Создание ответа аутентификации (пользователь не найден) (createAuthResponse\_whenUserNotFound\_shouldThrowBadCredentialsException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Создание ответа аутентификации (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с заданным email не найден, выбрасывается BadCredentialsException. |
| **Функция** | AuthServiceImpl.createAuthResponse |
| **Предусловия** | Мок UserService. userService.findByEmail возвращает Optional.empty(). |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | email (строка), ipAddress, deviceInfo, deviceId. |
| **Шаги выполнения** | 1. Настроить userService.findByEmail на возврат Optional.empty().   2. Вызвать authService.createAuthResponse внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается BadCredentialsException. |
| **Проверка** | Проверить тип выброшенного исключения. Убедиться, что вызывался только userService.findByEmail, а другие методы (updateLastLogin, generate..., mapTo...) нет. |
| **Полученный результат** | Функция выбросила ожидаемое исключение BadCredentialsException. Лишние вызовы зависимостей отсутствуют. |

* + 1. Успешное обновление токена (refreshToken\_whenTokenIsValid\_shouldReturnNewTokens)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное обновление токена |
| **Цель** | Проверить, что при валидном refresh токене: сессия находится, генерируется новая пара токенов (accessToken, refreshToken) и AuthResponse. |
| **Функция** | AuthServiceImpl.refreshToken |
| **Предусловия** | Моки UserSessionRepository (поиск сессии по токену возвращает Optional.of(session)), JwtTokenProvider (генерация новых токенов), UserService (маппинг в DTO). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | oldRefreshToken (строка), ipAddress, deviceInfo. |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.refreshToken(oldRefreshToken, ipAddress, deviceInfo).   3. Перехватить результат (AuthResponse). |
| **Ожидаемый результат** | Метод возвращает AuthResponse с новыми accessToken, refreshToken, expiresIn=360000000 и UserInfoDto. |
| **Проверка** | Проверить поля AuthResponse (новые токены, DTO пользователя). Проверить вызов sessionRepository.findByToken, jwtTokenProvider.generateAccessToken, jwtTokenProvider.generateRefreshToken, userService.mapToUserInfoDto. |
| **Полученный результат** | Функция успешно вернула AuthResponse с новыми токенами. Вызовы зависимостей корректны. |

* + 1. Успешный выход (токен существует) (logout\_whenTokenExists\_shouldDeleteToken)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешный выход (токен существует) |
| **Цель** | Проверить, что при выходе с существующим refresh токеном вызывается метод удаления сессии (deleteByToken) по этому токену. |
| **Функция** | AuthServiceImpl.logout |
| **Предусловия** | Мок UserSessionRepository. sessionRepository.findByToken возвращает Optional.of(session). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | refreshToken (строка). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.logout(refreshToken). |
| **Ожидаемый результат** | Вызываются sessionRepository.findByToken и sessionRepository.deleteByToken с переданным токеном. |
| **Проверка** | Проверить вызовы sessionRepository.findByToken(refreshToken) и sessionRepository.deleteByToken(refreshToken). |
| **Полученный результат** | Функция успешно вызвала удаление сессии для существующего токена. Вызовы зависимостей корректны. |

* + 1. Выход (токен не существует) (logout\_whenTokenDoesNotExist\_shouldNotDeleteToken)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Выход (токен не существует) |
| **Цель** | Проверить, что если refresh токен не найден (findByToken возвращает empty), метод удаления сессии (deleteByToken) не вызывается. |
| **Функция** | AuthServiceImpl.logout |
| **Предусловия** | Мок UserSessionRepository. sessionRepository.findByToken возвращает Optional.empty(). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | refreshToken (строка). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.logout(refreshToken). |
| **Ожидаемый результат** | Вызывается sessionRepository.findByToken, но не вызывается sessionRepository.deleteByToken. |
| **Проверка** | Проверить вызов sessionRepository.findByToken(refreshToken). Убедиться, что sessionRepository.deleteByToken не вызывался. |
| **Полученный результат** | Функция не вызвала удаление несуществующей сессии, как и ожидалось. Вызов findByToken корректен. |

* + 1. Успешная повторная отправка email подтверждения (resendVerificationEmail\_whenUserUnverified\_shouldSendEmail)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешная повторная отправка email подтверждения |
| **Цель** | Проверить, что для неверифицированного пользователя успешно отправляется новое письмо подтверждения с новым токеном (UUID). |
| **Функция** | AuthServiceImpl.resendVerificationEmail |
| **Предусловия** | Моки UserRepository (находит пользователя с isVerified=false), EmailService (для отправки письма). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | email (строка) неверифицированного пользователя. |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.resendVerificationEmail(email).   3. Захватить аргументы emailService.sendVerificationEmail. |
| **Ожидаемый результат** | Вызывается emailService.sendVerificationEmail с email, username и новым сгенерированным токеном (UUID). |
| **Проверка** | Проверить вызов userRepository.findByEmail. Проверить вызов emailService.sendVerificationEmail. С помощью stringCaptor проверить, что переданный токен не null и является валидным UUID. |
| **Полученный результат** | Функция успешно инициировала отправку нового письма подтверждения. Вызовы зависимостей корректны. |

* + 1. Повторная отправка email (пользователь не найден) (resendVerificationEmail\_whenUserNotFound\_shouldThrowNotFoundException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Повторная отправка email (пользователь не найден) |
| **Цель** | Проверить, что при попытке повторной отправки для несуществующего email выбрасывается ResponseStatusException (404 Not Found). |
| **Функция** | AuthServiceImpl.resendVerificationEmail |
| **Предусловия** | Мок UserRepository. userRepository.findByEmail возвращает Optional.empty(). |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | email (строка) несуществующего пользователя. |
| **Шаги выполнения** | 1. Настроить userRepository.findByEmail на возврат Optional.empty().   2. Вызвать authService.resendVerificationEmail внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 404 и сообщением "Пользователь не найден". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Убедиться, что emailService.sendVerificationEmail не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (404 Not Found). Email не отправлялся. |

* + 1. Повторная отправка email (пользователь уже верифицирован) (resendVerificationEmail\_whenUserAlreadyVerified\_shouldThrowBadRequestException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Повторная отправка email (пользователь уже верифицирован) |
| **Цель** | Проверить, что если email пользователя уже подтвержден, выбрасывается ResponseStatusException (400 Bad Request). |
| **Функция** | AuthServiceImpl.resendVerificationEmail |
| **Предусловия** | Мок UserRepository. userRepository.findByEmail возвращает пользователя с isVerified=true. |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | email (строка) верифицированного пользователя. |
| **Шаги выполнения** | 1. Настроить userRepository.findByEmail на возврат верифицированного пользователя.   2. Вызвать authService.resendVerificationEmail внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 400 и сообщением "Email уже подтвержден". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Убедиться, что emailService.sendVerificationEmail не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (400 Bad Request). Email не отправлялся. |

* + 1. Успешный запрос восстановления пароля (sendPasswordResetEmail\_whenUserExists\_shouldStoreCodeAndSendEmail)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешный запрос восстановления пароля |
| **Цель** | Проверить, что для существующего пользователя генерируется 6-значный код восстановления, сохраняется (storeResetCode) и отправляется email с этим кодом (sendPasswordResetEmail). |
| **Функция** | AuthServiceImpl.sendPasswordResetEmail |
| **Предусловия** | Моки UserRepository (находит пользователя по email), EmailService (для сохранения кода и отправки письма). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | email (строка) существующего пользователя. |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.sendPasswordResetEmail(email).   3. Захватить аргументы emailService.storeResetCode и emailService.sendPasswordResetEmail. |
| **Ожидаемый результат** | Генерируется 6-значный код. Вызываются emailService.storeResetCode и emailService.sendPasswordResetEmail с email, username и сгенерированным кодом. |
| **Проверка** | Проверить вызовы userRepository.findByEmail, emailService.storeResetCode, emailService.sendPasswordResetEmail. С помощью codeCaptor проверить, что код, переданный в storeResetCode и sendPasswordResetEmail, совпадает и является 6-значным числом. |
| **Полученный результат** | Функция успешно инициировала процесс восстановления пароля, сгенерировала, сохранила и отправила код. Вызовы зависимостей корректны. |

* + 1. Запрос восстановления пароля (пользователь не найден) (sendPasswordResetEmail\_whenUserNotFound\_shouldDoNothing)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Запрос восстановления пароля (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с указанным email не найден, никаких действий (сохранение кода, отправка email) не происходит и исключений не выбрасывается. |
| **Функция** | AuthServiceImpl.sendPasswordResetEmail |
| **Предусловия** | Мок UserRepository. userRepository.findByEmail возвращает Optional.empty(). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | email (строка) несуществующего пользователя. |
| **Шаги выполнения** | 1. Настроить userRepository.findByEmail на возврат Optional.empty().   2. Вызвать authService.sendPasswordResetEmail(email). |
| **Ожидаемый результат** | Метод завершается без ошибок. Никакие методы EmailService (storeResetCode, sendPasswordResetEmail) не вызываются. |
| **Проверка** | Проверить вызов userRepository.findByEmail. Убедиться, что emailService.storeResetCode и emailService.sendPasswordResetEmail не вызывались. |
| **Полученный результат** | Функция завершилась без ошибок и без отправки email/сохранения кода, как и ожидалось при отсутствии пользователя. |

* + 1. Успешная проверка кода восстановления (verifyPasswordResetCode\_whenCodeIsValid\_shouldStoreResetTokenAndReturnIt)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешная проверка кода восстановления |
| **Цель** | Проверить, что при валидном коде восстановления (verifyResetCode возвращает true): генерируется токен сброса (UUID), сохраняется (storeResetToken) и возвращается методом. |
| **Функция** | AuthServiceImpl.verifyPasswordResetCode |
| **Предусловия** | Мок EmailService. emailService.verifyResetCode возвращает true. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | email (строка), validCode (строка). |
| **Шаги выполнения** | 1. Настроить emailService.verifyResetCode на возврат true.   2. Вызвать authService.verifyPasswordResetCode(email, validCode).   3. Перехватить результат (resetToken) и захватить аргумент emailService.storeResetToken. |
| **Ожидаемый результат** | Метод возвращает resetToken (валидный UUID). Вызывается emailService.storeResetToken с этим же токеном. |
| **Проверка** | Проверить, что возвращенный токен не null и является валидным UUID. Проверить вызовы emailService.verifyResetCode и emailService.storeResetToken. С помощью resetTokenCaptor убедиться, что сохраненный токен совпадает с возвращенным. |
| **Полученный результат** | Функция успешно проверила код, вернула и сохранила resetToken. Вызовы зависимостей корректны. |

* + 1. Проверка невалидного кода восстановления (verifyPasswordResetCode\_whenCodeIsInvalid\_shouldThrowBadRequestException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Проверка невалидного кода восстановления |
| **Цель** | Проверить, что если код восстановления невалиден (verifyResetCode возвращает false), выбрасывается ResponseStatusException (400 Bad Request). |
| **Функция** | AuthServiceImpl.verifyPasswordResetCode |
| **Предусловия** | Мок EmailService. emailService.verifyResetCode возвращает false. |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | email (строка), invalidCode (строка). |
| **Шаги выполнения** | 1. Настроить emailService.verifyResetCode на возврат false.   2. Вызвать authService.verifyPasswordResetCode внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 400 и сообщением "Неверный или истекший код". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Убедиться, что emailService.storeResetToken не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (400 Bad Request). Токен сброса не генерировался/сохранялся. |

* + 1. Успешный сброс пароля (resetPassword\_whenTokenIsValidAndUserExists\_shouldUpdatePasswordAndInvalidateToken)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешный сброс пароля |
| **Цель** | Проверить, что при валидном токене сброса: находится пользователь по email из токена, его пароль обновляется на новый (зашифрованный), сохраняется, и токен сброса инвалидируется. |
| **Функция** | AuthServiceImpl.resetPassword |
| **Предусловия** | Моки EmailService (получение email по resetToken, инвалидация токена), UserRepository (поиск пользователя, сохранение), PasswordEncoder (кодирование нового пароля). userRepository.findByEmail находит пользователя. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | resetToken (строка, валидный UUID), newPassword (строка). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать authService.resetPassword(resetToken, newPassword).   3. Захватить аргумент userRepository.save. |
| **Ожидаемый результат** | Пароль пользователя (passwordHash) обновляется на зашифрованное значение newPassword. Обновляется метка updatedAt. Вызывается emailService.invalidateResetToken. |
| **Проверка** | Проверить вызовы emailService.getEmailByResetToken, userRepository.findByEmail, passwordEncoder.encode, userRepository.save, emailService.invalidateResetToken. С помощью userCaptor проверить, что сохраненный пароль (passwordHash) соответствует encodedPassword и updatedAt обновлен. |
| **Полученный результат** | Функция успешно обновила пароль пользователя и инвалидировала токен сброса. Вызовы зависимостей корректны. |

* + 1. Сброс пароля с невалидным токеном (resetPassword\_whenTokenIsInvalid\_shouldThrowBadRequestException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Сброс пароля с невалидным токеном |
| **Цель** | Проверить, что если токен сброса невалиден (getEmailByResetToken возвращает null), выбрасывается ResponseStatusException (400 Bad Request). |
| **Функция** | AuthServiceImpl.resetPassword |
| **Предусловия** | Мок EmailService. emailService.getEmailByResetToken возвращает null. |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | invalidResetToken (строка), newPassword (строка). |
| **Шаги выполнения** | 1. Настроить emailService.getEmailByResetToken на возврат null.   2. Вызвать authService.resetPassword внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 400 и сообщением "Неверный или истекший токен". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Убедиться, что userRepository.findByEmail, passwordEncoder.encode, userRepository.save, emailService.invalidateResetToken не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (400 Bad Request). Пароль не изменялся. |

* + 1. Сброс пароля (пользователь не найден) (resetPassword\_whenUserNotFound\_shouldThrowNotFoundException)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Сброс пароля (пользователь не найден) |
| **Цель** | Проверить, что если email, связанный с валидным токеном сброса, не найден в репозитории, выбрасывается ResponseStatusException (404 Not Found). |
| **Функция** | AuthServiceImpl.resetPassword |
| **Предусловия** | Моки EmailService (возвращает email по resetToken), UserRepository. userRepository.findByEmail возвращает Optional.empty(). |
| **Тип тестирования** | Позитивный (ожидаем корректную ошибку) |
| **Данные запроса** | resetToken (строка, валидный UUID), newPassword (строка). |
| **Шаги выполнения** | 1. Настроить emailService.getEmailByResetToken на возврат email.   2. Настроить userRepository.findByEmail на возврат Optional.empty().   3. Вызвать authService.resetPassword внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResponseStatusException со статусом 404 и сообщением "Пользователь не найден". |
| **Проверка** | Проверить тип и статус выброшенного исключения. Проверить вызовы emailService.getEmailByResetToken, userRepository.findByEmail. Убедиться, что passwordEncoder.encode, userRepository.save, emailService.invalidateResetToken не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResponseStatusException (404 Not Found). Пароль не изменялся. |

* + 1. Успешное создание анонимного токена (createAnonymousToken\_shouldGenerateAndReturnToken)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное создание анонимного токена |
| **Цель** | Проверить, что метод генерирует (generateAnonymousToken) и возвращает анонимный токен и фиксированный срок его действия (1800). |
| **Функция** | AuthServiceImpl.createAnonymousToken |
| **Предусловия** | Мок JwtTokenProvider. jwtTokenProvider.generateAnonymousToken возвращает expectedToken. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | deviceId (строка, опционально). |
| **Шаги выполнения** | 1. Настроить jwtTokenProvider.generateAnonymousToken.   2. Вызвать authService.createAnonymousToken(deviceId).   3. Перехватить результат (Map<String, Object>). |
| **Ожидаемый результат** | Метод возвращает Map, содержащую ключ anonymousToken со значением expectedToken (строка) и ключ expiresIn со значением 1800 (число). |
| **Проверка** | Проверить, что возвращенная карта содержит 2 элемента. Проверить значения anonymousToken и expiresIn. Проверить вызов jwtTokenProvider.generateAnonymousToken с переданным deviceId. |
| **Полученный результат** | Функция успешно вернула карту с анонимным токеном и сроком действия. Вызов jwtTokenProvider корректен. |

* 1. Сервис external
     1. Успешный поиск мест (searchPlaces\_shouldReturnPlaces)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешный поиск мест |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/search с валидным параметром query возвращает статус 200 OK и JSON с результатами поиска (PlaceSearchResponse). |
| **Функция** | PlaceController.searchPlaces (и мок PlaceService.searchPlaces) |
| **Предусловия** | Мок PlaceService настроен возвращать предопределенный PlaceSearchResponse при вызове searchPlaces. SecurityAutoConfiguration отключен для теста. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | GET-запрос на /api/v1/places/search с параметром query=test query. Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок placeService.searchPlaces на возврат mockSearchResponse.   2. Выполнить GET-запрос с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON, соответствующий mockSearchResponse (например, $.total=1, $.places[0].id="placeId1"). |
| **Проверка** | Проверка статус кода, Content-Type и структуры/значений JSON в теле ответа с помощью MockMvcResultMatchers. Неявная проверка вызова placeService.searchPlaces с ожидаемыми типами аргументов. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON, соответствующий ожидаемому PlaceSearchResponse. Вызов сервиса корректен. |

* + 1. Поиск мест без обязательного параметра (searchPlaces\_whenQueryMissing\_shouldReturnBadRequest)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Поиск мест без обязательного параметра query |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/search без обязательного параметра query возвращает статус 500 Internal Server Error (из-за стандартной обработки Spring при отсутствии явной валидации). |
| **Функция** | PlaceController.searchPlaces |
| **Предусловия** | SecurityAutoConfiguration отключен для теста. |
| **Тип тестирования** | Негативный (проверка обработки отсутствующего параметра) |
| **Данные запроса** | GET-запрос на /api/v1/places/search *без* параметра query. Accept: application/json. |
| **Шаги выполнения** | 1. Выполнить GET-запрос без параметра query с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 500. |
| **Проверка** | Проверка статус кода с помощью MockMvcResultMatchers. |
| **Полученный результат** | Запрос вернул статус 500 Internal Server Error, как и ожидалось при отсутствии параметра query по умолчанию в Spring. |

* + 1. Успешное получение деталей места (getPlaceDetails\_shouldReturnPlaceDetails)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение деталей места |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/{placeId} с валидным placeId возвращает статус 200 OK и JSON с деталями места (PlaceResponseDto). |
| **Функция** | PlaceController.getPlaceDetails (и мок PlaceService.getPlaceDetails) |
| **Предусловия** | Мок PlaceService настроен возвращать предопределенный PlaceResponseDto при вызове getPlaceDetails с указанным placeId. SecurityAutoConfiguration отключен. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | GET-запрос на /api/v1/places/{placeId} (где {placeId}="testPlaceId123"). Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок placeService.getPlaceDetails на возврат mockPlaceDetailsResponse.   2. Выполнить GET-запрос с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON, соответствующий mockPlaceDetailsResponse (например, $.id="testPlaceId123", $.name="Test Place"). |
| **Проверка** | Проверка статус кода, Content-Type и структуры/значений JSON в теле ответа. Неявная проверка вызова placeService.getPlaceDetails с правильным placeId. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON с деталями места. Вызов сервиса корректен. |

* + 1. Успешное автозаполнение мест (autocompletePlaces\_shouldReturnSuggestions)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное автозаполнение мест |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/autocomplete с параметром input возвращает статус 200 OK и JSON с предложениями (PlaceSuggestionResponse). |
| **Функция** | PlaceController.autocompletePlaces (и мок PlaceService.autocompletePlaces) |
| **Предусловия** | Мок PlaceService настроен возвращать предопределенный PlaceSuggestionResponse при вызове autocompletePlaces. SecurityAutoConfiguration отключен. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | GET-запрос на /api/v1/places/autocomplete с параметром input=test input. Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок placeService.autocompletePlaces.   2. Выполнить GET-запрос с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON с массивом suggestions, соответствующий mockSuggestionResponse. |
| **Проверка** | Проверка статус кода, Content-Type и структуры/значений JSON (например, $.suggestions[0].id). Неявная проверка вызова placeService.autocompletePlaces. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON с предложениями для автозаполнения. Вызов сервиса корректен. |

* + 1. Автозаполнение мест без обязательного параметра (autocompletePlaces\_whenInputMissing\_shouldReturnBadRequest)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Автозаполнение мест без обязательного параметра input |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/autocomplete без обязательного параметра input возвращает статус 500 Internal Server Error. |
| **Функция** | PlaceController.autocompletePlaces |
| **Предусловия** | SecurityAutoConfiguration отключен для теста. |
| **Тип тестирования** | Негативный (проверка обработки отсутствующего параметра) |
| **Данные запроса** | GET-запрос на /api/v1/places/autocomplete *без* параметра input. Accept: application/json. |
| **Шаги выполнения** | 1. Выполнить GET-запрос без параметра input с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 500. |
| **Проверка** | Проверка статус кода с помощью MockMvcResultMatchers. |
| **Полученный результат** | Запрос вернул статус 500 Internal Server Error, как и ожидалось при отсутствии параметра input. |

* + 1. Успешный поиск мест поблизости (getNearbyPlaces\_shouldReturnNearbyPlaces)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешный поиск мест поблизости |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/nearby с обязательными параметрами lat и lon возвращает статус 200 OK и JSON с найденными местами (PlaceSearchResponse). |
| **Функция** | PlaceController.getNearbyPlaces (и мок PlaceService.getNearbyPlaces) |
| **Предусловия** | Мок PlaceService настроен возвращать предопределенный PlaceSearchResponse при вызове getNearbyPlaces. SecurityAutoConfiguration отключен. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | GET-запрос на /api/v1/places/nearby с параметрами lat=55.75 и lon=37.62. Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок placeService.getNearbyPlaces.   2. Выполнить GET-запрос с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON, соответствующий nearbyResponse (например, $.total=1, $.places[0].name="Place 1"). |
| **Проверка** | Проверка статус кода, Content-Type и структуры/значений JSON. Неявная проверка вызова placeService.getNearbyPlaces с double-аргументами. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON с местами поблизости. Вызов сервиса корректен. |

* + 1. Поиск мест поблизости без обязательных параметров (getNearbyPlaces\_whenLatLonMissing\_shouldReturnBadRequest)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Поиск мест поблизости без обязательных параметров lat/lon |
| **Цель** | Проверить, что GET-запрос к /api/v1/places/nearby без обязательных параметров lat и lon возвращает статус 500 Internal Server Error. |
| **Функция** | PlaceController.getNearbyPlaces |
| **Предусловия** | SecurityAutoConfiguration отключен для теста. |
| **Тип тестирования** | Негативный (проверка обработки отсутствующих параметров) |
| **Данные запроса** | GET-запрос на /api/v1/places/nearby *без* параметров lat и lon. Accept: application/json. |
| **Шаги выполнения** | 1. Выполнить GET-запрос без параметров lat/lon с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 500. |
| **Проверка** | Проверка статус кода с помощью MockMvcResultMatchers. |
| **Полученный результат** | Запрос вернул статус 500 Internal Server Error, как и ожидалось при отсутствии параметров lat/lon. |

* + 1. Успешная генерация списка вещей (generatePackingList\_shouldReturnPackingList)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешная генерация списка вещей |
| **Цель** | Проверить, что POST-запрос к /api/v1/ai/packing-list с валидным телом (PackingListRequest) возвращает статус 200 OK и JSON с сгенерированным списком (PackingListResponse). |
| **Функция** | AiController.generatePackingList (и мок AiService.generatePackingList) |
| **Предусловия** | Мок AiService настроен возвращать предопределенный PackingListResponse при вызове generatePackingList. SecurityAutoConfiguration отключен. ObjectMapper доступен. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | POST-запрос на /api/v1/ai/packing-list. Content-Type: application/json. Тело: JSON, соответствующий validPackingListRequest. Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок aiService.generatePackingList.   2. Выполнить POST-запрос с JSON-телом с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON, соответствующий mockPackingListResponse (например, $.totalItems=0). |
| **Проверка** | Проверка статус кода, Content-Type и структуры/значений JSON. Неявная проверка вызова aiService.generatePackingList с объектом PackingListRequest. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON сгенерированного списка. Вызов сервиса корректен. |

* + 1. Генерация списка вещей без тела запроса (generatePackingList\_whenBodyMissing\_shouldReturnBadRequest)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Генерация списка вещей без тела запроса |
| **Цель** | Проверить, что POST-запрос к /api/v1/ai/packing-list без тела запроса возвращает статус 500 Internal Server Error (из-за стандартной обработки Spring). |
| **Функция** | AiController.generatePackingList |
| **Предусловия** | SecurityAutoConfiguration отключен для теста. |
| **Тип тестирования** | Негативный (проверка обработки отсутствующего тела) |
| **Данные запроса** | POST-запрос на /api/v1/ai/packing-list *без* тела запроса. Content-Type: application/json. Accept: application/json. |
| **Шаги выполнения** | 1. Выполнить POST-запрос без тела с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 500. |
| **Проверка** | Проверка статус кода с помощью MockMvcResultMatchers. |
| **Полученный результат** | Запрос вернул статус 500 Internal Server Error, как и ожидалось при отсутствии тела запроса. |

* + 1. Генерация списка вещей с невалидным телом (generatePackingList\_whenBodyInvalid\_shouldReturnBadRequest)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Генерация списка вещей с невалидным телом |
| **Цель** | Проверить, что POST-запрос к /api/v1/ai/packing-list с телом, не соответствующим PackingListRequest (например, пропущено обязательное поле destination), **в текущей реализации теста** возвращает 200 OK (т.к. валидация не проверяется/не настроена). |
| **Функция** | AiController.generatePackingList |
| **Предусловия** | SecurityAutoConfiguration отключен. ObjectMapper доступен. **Важно:** В предоставленном коде нет активной проверки на BadRequest для невалидного тела, поэтому тест не ожидает 400. |
| **Тип тестирования** | Негативный (но текущий код теста не проверяет ошибку) |
| **Данные запроса** | POST-запрос на /api/v1/ai/packing-list. Content-Type: application/json. Тело: JSON с пропущенным полем destination. Accept: application/json. |
| **Шаги выполнения** | 1. Создать невалидный invalidRequest.   2. Выполнить POST-запрос с этим телом с помощью MockMvc. |
| **Ожидаемый результат** | **В текущей реализации теста:** Статус код 200. (Комментарий в коде указывает, что ожидался бы 400 при наличии @Valid). |
| **Проверка** | Проверка только статус кода (на 200 OK, согласно текущему коду). |
| **Полученный результат** | Запрос вернул статус 200 OK, так как валидация не активна в тесте/коде. |

* + 1. Успешное получение шаблонов списков вещей (getPackingListTemplates\_shouldReturnTemplates)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение шаблонов списков вещей |
| **Цель** | Проверить, что GET-запрос к /api/v1/ai/packing-list/templates возвращает статус 200 OK и JSON со списком шаблонов (PackingListTemplatesResponse). |
| **Функция** | AiController.getPackingListTemplates (и мок AiService.getPackingListTemplates) |
| **Предусловия** | Мок AiService настроен возвращать предопределенный PackingListTemplatesResponse при вызове getPackingListTemplates. SecurityAutoConfiguration отключен. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | GET-запрос на /api/v1/ai/packing-list/templates. Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок aiService.getPackingListTemplates.   2. Выполнить GET-запрос с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON с массивом templates. |
| **Проверка** | Проверка статус кода, Content-Type и наличия массива templates. Неявная проверка вызова aiService.getPackingListTemplates. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON со списком шаблонов. Вызов сервиса корректен. |

* + 1. Успешное получение содержимого шаблона (getPackingListTemplateContent\_shouldReturnTemplateContent)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение содержимого шаблона |
| **Цель** | Проверить, что GET-запрос к /api/v1/ai/packing-list/template/{templateId} возвращает статус 200 OK и JSON с содержимым шаблона (PackingListTemplateContent). |
| **Функция** | AiController.getPackingListTemplateContent (и мок AiService.getPackingListTemplateContent) |
| **Предусловия** | Мок AiService настроен возвращать предопределенный PackingListTemplateContent при вызове getPackingListTemplateContent. SecurityAutoConfiguration отключен. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | GET-запрос на /api/v1/ai/packing-list/template/{templateId} (где {templateId}="template1"). Accept: application/json. |
| **Шаги выполнения** | 1. Настроить мок aiService.getPackingListTemplateContent.   2. Выполнить GET-запрос с помощью MockMvc. |
| **Ожидаемый результат** | Статус код 200. Content-Type: application/json. Тело ответа содержит JSON, соответствующий mockTemplateContentResponse (например, $.id="template1", $.name="Summer City Trip"). |
| **Проверка** | Проверка статус кода, Content-Type и структуры/значений JSON. Неявная проверка вызова aiService.getPackingListTemplateContent с правильным templateId. |
| **Полученный результат** | Запрос вернул статус 200 OK и JSON с содержимым шаблона. Вызов сервиса корректен. |

* 1. Сервис library
     1. Успешное получение опубликованных маршрутов (testGetPublishedRoutes\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение страницы опубликованных маршрутов |
| **Цель** | Проверить, что метод getPublishedRoutes успешно возвращает страницу (Page) DTO (RoutePreviewDto) одобренных опубликованных маршрутов. |
| **Функция** | LibraryService.getPublishedRoutes |
| **Предусловия** | Мок PublishedRouteRepository. findAllApproved возвращает Page<PublishedRoute>. Мок MapperService. toRoutePreviewDto преобразует PublishedRoute в RoutePreviewDto. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | Pageable объект. |
| **Шаги выполнения** | 1. Настроить моки publishedRouteRepository.findAllApproved и mapperService.toRoutePreviewDto.   2. Вызвать libraryService.getPublishedRoutes(pageable).   3. Перехватить результат (Page<RoutePreviewDto>). |
| **Ожидаемый результат** | Метод возвращает Page<RoutePreviewDto> с данными, соответствующими мокам. Содержимое страницы состоит из DTO, полученных от MapperService. |
| **Проверка** | Проверка, что результат не null, проверка размера содержимого и общего количества элементов. Проверка вызова publishedRouteRepository.findAllApproved с переданным pageable. Проверка вызова mapperService.toRoutePreviewDto для каждого элемента из мока репозитория. |
| **Полученный результат** | Функция успешно вернула страницу DTO опубликованных маршрутов. Вызовы зависимостей корректны. |

* + 1. Успешное получение деталей маршрута (testGetRouteDetails\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение деталей опубликованного маршрута |
| **Цель** | Проверить, что метод getRouteDetails находит одобренный маршрут по ID, увеличивает счетчик просмотров (viewCount), сохраняет изменения и возвращает DTO деталей (PublicRouteDetailDto). |
| **Функция** | LibraryService.getRouteDetails |
| **Предусловия** | Мок PublishedRouteRepository. findByIdAndIsApprovedTrue возвращает Optional.of(publishedRoute). save возвращает обновленный маршрут. Мок MapperService. toPublicRouteDetailDto преобразует PublishedRoute в PublicRouteDetailDto. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long). |
| **Шаги выполнения** | 1. Настроить моки publishedRouteRepository.findByIdAndIsApprovedTrue, publishedRouteRepository.save, mapperService.toPublicRouteDetailDto.   2. Вызвать libraryService.getRouteDetails(routeId).   3. Перехватить результат (PublicRouteDetailDto). |
| **Ожидаемый результат** | Метод возвращает PublicRouteDetailDto, соответствующий найденному маршруту. Счетчик просмотров (viewCount) у объекта publishedRoute увеличен на 1 перед сохранением. |
| **Проверка** | Проверка, что результат не null и соответствует DTO из мока маппера. Проверка вызова publishedRouteRepository.findByIdAndIsApprovedTrue с routeId. Проверка вызова publishedRouteRepository.save с объектом publishedRoute, у которого viewCount увеличен. Проверка вызова mapperService.toPublicRouteDetailDto. |
| **Полученный результат** | Функция успешно вернула детали маршрута и увеличила счетчик просмотров. Вызовы зависимостей корректны. |

* + 1. Получение деталей маршрута (не найден) (testGetRouteDetails\_NotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение деталей опубликованного маршрута (не найден) |
| **Цель** | Проверить, что если одобренный маршрут по ID не найден, метод getRouteDetails выбрасывает ResourceNotFoundException. |
| **Функция** | LibraryService.getRouteDetails |
| **Предусловия** | Мок PublishedRouteRepository. findByIdAndIsApprovedTrue возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long) несуществующего/неодобренного маршрута. |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.findByIdAndIsApprovedTrue на возврат Optional.empty().   2. Вызвать libraryService.getRouteDetails(routeId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Route not found with id {routeId}". |
| **Проверка** | Проверка типа и сообщения выброшенного исключения. Проверка вызова publishedRouteRepository.findByIdAndIsApprovedTrue с routeId. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Успешная публикация маршрута (testPublishRoute\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешная публикация нового маршрута |
| **Цель** | Проверить, что метод publishRoute успешно создает и сохраняет новую сущность PublishedRoute, если маршрут с таким originalRouteId еще не опубликован и пользователь существует, и возвращает DTO (PublicRouteDto). |
| **Функция** | LibraryService.publishRoute |
| **Предусловия** | Мок PublishedRouteRepository. existsByOriginalRouteId возвращает false. save возвращает сохраненный PublishedRoute с ID. Мок UserRepository. findById возвращает Optional.of(user). Мок MapperService. toPublicRouteDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | trip (Trip), userId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.publishRoute(trip, userId).   3. Перехватить результат (PublicRouteDto) и захватить аргумент publishedRouteRepository.save. |
| **Ожидаемый результат** | Метод возвращает PublicRouteDto. Новая сущность PublishedRoute сохраняется с данными из trip и userId, isApproved=false, viewCount=0. |
| **Проверка** | Проверка, что результат не null и соответствует DTO из мока маппера. Проверка вызовов publishedRouteRepository.existsByOriginalRouteId, userRepository.findById, publishedRouteRepository.save, mapperService.toPublicRouteDto. Проверка полей сохраненной сущности (originalRouteId, userId, title и т.д.). |
| **Полученный результат** | Функция успешно создала и сохранила опубликованный маршрут. Вызовы зависимостей корректны. |

* + 1. Публикация маршрута (пользователь не найден) (testPublishRoute\_UserNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Публикация маршрута (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с указанным userId не найден, метод publishRoute выбрасывает ResourceNotFoundException. |
| **Функция** | LibraryService.publishRoute |
| **Предусловия** | Мок PublishedRouteRepository. existsByOriginalRouteId возвращает false. Мок UserRepository. findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | trip (Trip), nonExistentUserId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.publishRoute(trip, nonExistentUserId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "User not found with id {userId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызовов publishedRouteRepository.existsByOriginalRouteId, userRepository.findById. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Публикация маршрута (уже существует) (testPublishRoute\_AlreadyExists)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Публикация маршрута (уже существует) |
| **Цель** | Проверить, что если маршрут с таким originalRouteId уже опубликован, метод publishRoute выбрасывает IllegalStateException. |
| **Функция** | LibraryService.publishRoute |
| **Предусловия** | Мок PublishedRouteRepository. existsByOriginalRouteId возвращает true. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | trip (Trip), userId (Long). |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.existsByOriginalRouteId на возврат true.   2. Вызвать libraryService.publishRoute(trip, userId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается IllegalStateException с сообщением "Route is already published". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова publishedRouteRepository.existsByOriginalRouteId. |
| **Полученный результат** | Функция выбросила ожидаемое исключение IllegalStateException. |

* + 1. Успешное одобрение маршрута (testApprovePublishedRoute\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное одобрение опубликованного маршрута |
| **Цель** | Проверить, что метод approvePublishedRoute находит маршрут по ID, устанавливает флаг isApproved в true, сохраняет и возвращает DTO. |
| **Функция** | LibraryService.approvePublishedRoute |
| **Предусловия** | Мок PublishedRouteRepository. findById возвращает Optional.of(publishedRoute). save возвращает обновленный маршрут. Мок MapperService. toPublicRouteDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long). |
| **Шаги выполнения** | 1. Настроить моки. Убедиться, что у publishedRoute isApproved=false.   2. Вызвать libraryService.approvePublishedRoute(routeId).   3. Перехватить результат (PublicRouteDto). |
| **Ожидаемый результат** | Метод возвращает PublicRouteDto. Флаг isApproved у сущности publishedRoute устанавливается в true перед сохранением. |
| **Проверка** | Проверка, что результат не null. Проверка, что publishedRoute.getIsApproved() стал true. Проверка вызовов publishedRouteRepository.findById, publishedRouteRepository.save, mapperService.toPublicRouteDto. |
| **Полученный результат** | Функция успешно одобрила маршрут и вернула DTO. Вызовы зависимостей корректны. |

* + 1. Одобрение маршрута (не найден) (testApprovePublishedRoute\_NotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Одобрение опубликованного маршрута (не найден) |
| **Цель** | Проверить, что если маршрут по ID не найден, метод approvePublishedRoute выбрасывает ResourceNotFoundException. |
| **Функция** | LibraryService.approvePublishedRoute |
| **Предусловия** | Мок PublishedRouteRepository. findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long) несуществующего маршрута. |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.findById на возврат Optional.empty().   2. Вызвать libraryService.approvePublishedRoute(routeId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Published route not found with id {routeId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова publishedRouteRepository.findById. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Успешное удаление маршрута (testDeletePublishedRoute\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное удаление опубликованного маршрута |
| **Цель** | Проверить, что метод deletePublishedRoute находит маршрут по ID и вызывает метод delete репозитория для найденного маршрута. |
| **Функция** | LibraryService.deletePublishedRoute |
| **Предусловия** | Мок PublishedRouteRepository. findById возвращает Optional.of(publishedRoute). delete ничего не возвращает. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long) существующего маршрута. |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.findById.   2. Вызвать libraryService.deletePublishedRoute(routeId). |
| **Ожидаемый результат** | Метод успешно завершается. Вызывается publishedRouteRepository.delete с объектом publishedRoute. |
| **Проверка** | Проверка вызовов publishedRouteRepository.findById и publishedRouteRepository.delete. |
| **Полученный результат** | Функция успешно вызвала удаление найденного маршрута. Вызовы зависимостей корректны. |

* + 1. Удаление маршрута (не найден) (testDeletePublishedRoute\_NotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Удаление опубликованного маршрута (не найден) |
| **Цель** | Проверить, что если маршрут по ID не найден, метод deletePublishedRoute выбрасывает ResourceNotFoundException. |
| **Функция** | LibraryService.deletePublishedRoute |
| **Предусловия** | Мок PublishedRouteRepository. findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long) несуществующего маршрута. |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.findById на возврат Optional.empty().   2. Вызвать libraryService.deletePublishedRoute(routeId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Published route not found with id {routeId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова publishedRouteRepository.findById. Убедиться, что delete не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. Удаление не производилось. |

* + 1. Успешный поиск маршрутов (testSearchRoutes\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешный поиск маршрутов по строке запроса |
| **Цель** | Проверить, что метод searchRoutes вызывает publishedRouteRepository.searchByQuery и возвращает страницу DTO (RoutePreviewDto). |
| **Функция** | LibraryService.searchRoutes |
| **Предусловия** | Мок PublishedRouteRepository. searchByQuery возвращает Page<PublishedRoute>. Мок MapperService. toRoutePreviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | query (String), pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.searchRoutes(query, pageable).   3. Перехватить результат (Page<RoutePreviewDto>). |
| **Ожидаемый результат** | Метод возвращает Page<RoutePreviewDto> с результатами поиска. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызова publishedRouteRepository.searchByQuery с query и pageable. Проверка вызова mapperService.toRoutePreviewDto. |
| **Полученный результат** | Функция успешно вернула страницу результатов поиска. Вызовы зависимостей корректны. |

* + 1. Успешное получение отфильтрованных маршрутов (testGetFilteredRoutes\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение отфильтрованных маршрутов |
| **Цель** | Проверить, что метод getFilteredRoutes вызывает publishedRouteRepository.findWithFilters с переданными параметрами и возвращает страницу DTO (RoutePreviewDto). |
| **Функция** | LibraryService.getFilteredRoutes |
| **Предусловия** | Мок PublishedRouteRepository. findWithFilters возвращает Page<PublishedRoute>. Мок MapperService. toRoutePreviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | country, city, durationMin, durationMax, tag (String, Integer), pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.getFilteredRoutes(country, city, durationMin, durationMax, tag, pageable).   3. Перехватить результат (Page<RoutePreviewDto>). |
| **Ожидаемый результат** | Метод возвращает Page<RoutePreviewDto> с отфильтрованными результатами. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызова publishedRouteRepository.findWithFilters с корректными параметрами. Проверка вызова mapperService.toRoutePreviewDto. |
| **Полученный результат** | Функция успешно вернула страницу отфильтрованных маршрутов. Вызовы зависимостей корректны. |

* + 1. Успешное получение популярных маршрутов (testGetPopularRoutes\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение популярных маршрутов |
| **Цель** | Проверить, что метод getPopularRoutes вызывает publishedRouteRepository.findPopular и возвращает страницу DTO (RoutePreviewDto). |
| **Функция** | LibraryService.getPopularRoutes |
| **Предусловия** | Мок PublishedRouteRepository. findPopular возвращает Page<PublishedRoute>. Мок MapperService. toRoutePreviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.getPopularRoutes(pageable).   3. Перехватить результат (Page<RoutePreviewDto>). |
| **Ожидаемый результат** | Метод возвращает Page<RoutePreviewDto> с популярными маршрутами. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызова publishedRouteRepository.findPopular с pageable. Проверка вызова mapperService.toRoutePreviewDto. |
| **Полученный результат** | Функция успешно вернула страницу популярных маршрутов. Вызовы зависимостей корректны. |

* + 1. Успешное получение маршрутов с наивысшим рейтингом (testGetMostRatedRoutes\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение маршрутов с наивысшим рейтингом |
| **Цель** | Проверить, что метод getMostRatedRoutes вызывает publishedRouteRepository.findMostRated и возвращает страницу DTO (RoutePreviewDto). |
| **Функция** | LibraryService.getMostRatedRoutes |
| **Предусловия** | Мок PublishedRouteRepository. findMostRated возвращает Page<PublishedRoute>. Мок MapperService. toRoutePreviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.getMostRatedRoutes(pageable).   3. Перехватить результат (Page<RoutePreviewDto>). |
| **Ожидаемый результат** | Метод возвращает Page<RoutePreviewDto> с маршрутами с наивысшим рейтингом. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызова publishedRouteRepository.findMostRated с pageable. Проверка вызова mapperService.toRoutePreviewDto. |
| **Полученный результат** | Функция успешно вернула страницу маршрутов с наивысшим рейтингом. Вызовы зависимостей корректны. |

* + 1. Успешное получение опубликованных маршрутов пользователя (testGetUserPublishedRoutes\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение опубликованных маршрутов пользователя |
| **Цель** | Проверить, что метод getUserPublishedRoutes находит пользователя по ID, запрашивает его одобренные опубликованные маршруты и возвращает список DTO (RoutePreviewDto). |
| **Функция** | LibraryService.getUserPublishedRoutes |
| **Предусловия** | Мок UserRepository. findById возвращает Optional.of(user). Мок PublishedRouteRepository. findByUserIdAndIsApprovedTrue возвращает List<PublishedRoute>. Мок MapperService. toRoutePreviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать libraryService.getUserPublishedRoutes(userId).   3. Перехватить результат (List<RoutePreviewDto>). |
| **Ожидаемый результат** | Метод возвращает List<RoutePreviewDto> с маршрутами указанного пользователя. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызовов userRepository.findById, publishedRouteRepository.findByUserIdAndIsApprovedTrue, mapperService.toRoutePreviewDto. |
| **Полученный результат** | Функция успешно вернула список опубликованных маршрутов пользователя. Вызовы зависимостей корректны. |

* + 1. Получение опубликованных маршрутов пользователя (пользователь не найден) (testGetUserPublishedRoutes\_UserNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение опубликованных маршрутов пользователя (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с указанным userId не найден, метод getUserPublishedRoutes выбрасывает ResourceNotFoundException. |
| **Функция** | LibraryService.getUserPublishedRoutes |
| **Предусловия** | Мок UserRepository. findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | nonExistentUserId (Long). |
| **Шаги выполнения** | 1. Настроить userRepository.findById на возврат Optional.empty().   2. Вызвать libraryService.getUserPublishedRoutes(nonExistentUserId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "User not found with id {userId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова userRepository.findById. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Успешное добавление нового отзыва (testAddOrUpdateReview\_AddNew\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное добавление нового отзыва |
| **Цель** | Проверить, что если отзыв для маршрута от пользователя не существует, addOrUpdateReview создает и сохраняет новую сущность RouteRating и возвращает DTO (ReviewDto). |
| **Функция** | ReviewService.addOrUpdateReview |
| **Предусловия** | Мок PublishedRouteRepository. findByIdAndIsApprovedTrue находит маршрут. Мок UserRepository. existsById возвращает true. Мок RouteRatingRepository. findByPublishedRouteIdAndUserId возвращает Optional.empty(). save возвращает сохраненный RouteRating с ID. Мок MapperService. toReviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long), userId (Long), newRating (Integer), newComment (String). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.addOrUpdateReview(routeId, userId, newRating, newComment).   3. Перехватить результат (ReviewDto) и захватить аргумент ratingRepository.save. |
| **Ожидаемый результат** | Метод возвращает ReviewDto. Новая сущность RouteRating сохраняется с переданными routeId, userId, newRating. |
| **Проверка** | Проверка, что результат не null и соответствует DTO из мока маппера. Проверка вызовов репозиториев (findByIdAndIsApprovedTrue, existsById, findBy..., save). С помощью ratingCaptor проверить поля сохраненной сущности. |
| **Полученный результат** | Функция успешно добавила новый отзыв. Вызовы зависимостей корректны. |

* + 1. Успешное обновление существующего отзыва (testAddOrUpdateReview\_UpdateExisting\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное обновление существующего отзыва |
| **Цель** | Проверить, что если отзыв для маршрута от пользователя существует, addOrUpdateReview обновляет рейтинг и комментарий существующей сущности RouteRating, сохраняет ее и возвращает DTO. |
| **Функция** | ReviewService.addOrUpdateReview |
| **Предусловия** | Мок PublishedRouteRepository. findByIdAndIsApprovedTrue находит маршрут. Мок UserRepository. existsById возвращает true. Мок RouteRatingRepository. findByPublishedRouteIdAndUserId возвращает Optional.of(routeRating). save возвращает обновленный RouteRating. Мок MapperService. toReviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long), userId (Long), newRating (Integer), newComment (String). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.addOrUpdateReview(routeId, userId, newRating, newComment).   3. Перехватить результат (ReviewDto). |
| **Ожидаемый результат** | Метод возвращает ReviewDto. Поля rating и comment у существующей сущности routeRating обновляются перед сохранением. |
| **Проверка** | Проверка, что результат не null и соответствует DTO. Проверка, что поля у объекта routeRating изменились на newRating и newComment. Проверка вызовов репозиториев (findByIdAndIsApprovedTrue, existsById, findBy..., save). |
| **Полученный результат** | Функция успешно обновила существующий отзыв. Вызовы зависимостей корректны. |

* + 1. Добавление/обновление отзыва (маршрут не найден) (testAddOrUpdateReview\_RouteNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Добавление/обновление отзыва (маршрут не найден) |
| **Цель** | Проверить, что если одобренный маршрут по ID не найден, метод addOrUpdateReview выбрасывает ResourceNotFoundException. |
| **Функция** | ReviewService.addOrUpdateReview |
| **Предусловия** | Мок PublishedRouteRepository. findByIdAndIsApprovedTrue возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long), userId (Long), rating (Integer), comment (String). |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.findByIdAndIsApprovedTrue на возврат Optional.empty().   2. Вызвать reviewService.addOrUpdateReview(...) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Route not found with id {routeId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова publishedRouteRepository.findByIdAndIsApprovedTrue. Убедиться, что другие репозитории не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Добавление/обновление отзыва (пользователь не найден) (testAddOrUpdateReview\_UserNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Добавление/обновление отзыва (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с userId не существует, метод addOrUpdateReview выбрасывает ResourceNotFoundException. |
| **Функция** | ReviewService.addOrUpdateReview |
| **Предусловия** | Мок PublishedRouteRepository. findByIdAndIsApprovedTrue находит маршрут. Мок UserRepository. existsById возвращает false. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long), userId (Long), rating (Integer), comment (String). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.addOrUpdateReview(...) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "User not found with id {userId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызовов publishedRouteRepository.findByIdAndIsApprovedTrue, userRepository.existsById. Убедиться, что RouteRatingRepository не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Успешное удаление отзыва (testDeleteReview\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное удаление отзыва |
| **Цель** | Проверить, что метод deleteReview успешно удаляет отзыв, если и маршрут, и сам отзыв существуют. |
| **Функция** | ReviewService.deleteReview |
| **Предусловия** | Мок PublishedRouteRepository. existsById возвращает true. Мок RouteRatingRepository. existsByPublishedRouteIdAndUserId возвращает true. deleteByPublishedRouteIdAndUserId ничего не делает. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long), userId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.deleteReview(routeId, userId). |
| **Ожидаемый результат** | Метод успешно завершается без исключений. Вызывается ratingRepository.deleteByPublishedRouteIdAndUserId. |
| **Проверка** | Проверка вызовов publishedRouteRepository.existsById, ratingRepository.existsBy..., ratingRepository.deleteBy.... |
| **Полученный результат** | Функция успешно вызвала удаление отзыва. Вызовы зависимостей корректны. |

* + 1. Удаление отзыва (маршрут не найден) (testDeleteReview\_RouteNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Удаление отзыва (маршрут не найден) |
| **Цель** | Проверить, что если маршрут с routeId не найден, метод deleteReview выбрасывает ResourceNotFoundException. |
| **Функция** | ReviewService.deleteReview |
| **Предусловия** | Мок PublishedRouteRepository. existsById возвращает false. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long), userId (Long). |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.existsById на возврат false.   2. Вызвать reviewService.deleteReview(...) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Route not found with id {routeId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова publishedRouteRepository.existsById. Убедиться, что RouteRatingRepository не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Удаление отзыва (отзыв не найден) (testDeleteReview\_ReviewNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Удаление отзыва (отзыв не найден) |
| **Цель** | Проверить, что если отзыв для пары routeId и userId не найден, метод deleteReview выбрасывает ResourceNotFoundException. |
| **Функция** | ReviewService.deleteReview |
| **Предусловия** | Мок PublishedRouteRepository. existsById возвращает true. Мок RouteRatingRepository. existsByPublishedRouteIdAndUserId возвращает false. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long), userId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.deleteReview(...) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Review not found for route {routeId} and user {userId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызовов publishedRouteRepository.existsById, ratingRepository.existsBy.... Убедиться, что deleteBy... не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Успешное получение отзывов для маршрута (testGetRouteReviews\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение отзывов для маршрута |
| **Цель** | Проверить, что метод getRouteReviews находит маршрут, запрашивает страницу отзывов для него и возвращает страницу DTO (ReviewDto). |
| **Функция** | ReviewService.getRouteReviews |
| **Предусловия** | Мок PublishedRouteRepository. existsById возвращает true. Мок RouteRatingRepository. findByPublishedRouteId возвращает Page<RouteRating>. Мок MapperService. toReviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long), pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.getRouteReviews(routeId, pageable).   3. Перехватить результат (Page<ReviewDto>). |
| **Ожидаемый результат** | Метод возвращает Page<ReviewDto> с отзывами для указанного маршрута. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызовов publishedRouteRepository.existsById, ratingRepository.findByPublishedRouteId, mapperService.toReviewDto. |
| **Полученный результат** | Функция успешно вернула страницу отзывов для маршрута. Вызовы зависимостей корректны. |

* + 1. Получение отзывов для маршрута (маршрут не найден) (testGetRouteReviews\_RouteNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение отзывов для маршрута (маршрут не найден) |
| **Цель** | Проверить, что если маршрут с routeId не найден, метод getRouteReviews выбрасывает ResourceNotFoundException. |
| **Функция** | ReviewService.getRouteReviews |
| **Предусловия** | Мок PublishedRouteRepository. existsById возвращает false. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long), pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить publishedRouteRepository.existsById на возврат false.   2. Вызвать reviewService.getRouteReviews(...) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Route not found with id {routeId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова publishedRouteRepository.existsById. Убедиться, что RouteRatingRepository не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Успешное получение отзыва пользователя (testGetUserReview\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение отзыва пользователя для маршрута |
| **Цель** | Проверить, что метод getUserReview находит отзыв по routeId и userId и возвращает DTO (ReviewDto). |
| **Функция** | ReviewService.getUserReview |
| **Предусловия** | Мок RouteRatingRepository. findByPublishedRouteIdAndUserId возвращает Optional.of(routeRating). Мок MapperService. toReviewDto преобразует в DTO. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | routeId (Long), userId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать reviewService.getUserReview(routeId, userId).   3. Перехватить результат (ReviewDto). |
| **Ожидаемый результат** | Метод возвращает ReviewDto, соответствующий найденному отзыву. |
| **Проверка** | Проверка, что результат не null и соответствует DTO из мока маппера. Проверка вызова ratingRepository.findByPublishedRouteIdAndUserId. Проверка вызова mapperService.toReviewDto. |
| **Полученный результат** | Функция успешно вернула DTO отзыва пользователя. Вызовы зависимостей корректны. |

* + 1. Получение отзыва пользователя (отзыв не найден) (testGetUserReview\_ReviewNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение отзыва пользователя (отзыв не найден) |
| **Цель** | Проверить, что если отзыв для пары routeId и userId не найден, метод getUserReview выбрасывает ResourceNotFoundException. |
| **Функция** | ReviewService.getUserReview |
| **Предусловия** | Мок RouteRatingRepository. findByPublishedRouteIdAndUserId возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | routeId (Long), userId (Long). |
| **Шаги выполнения** | 1. Настроить ratingRepository.findByPublishedRouteIdAndUserId на возврат Optional.empty().   2. Вызвать reviewService.getUserReview(...) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException с сообщением "Review not found for route {routeId} and user {userId}". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызова ratingRepository.findByPublishedRouteIdAndUserId. Убедиться, что MapperService не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* 1. Сервис planner
     1. Успешное создание поездки (createTrip\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное создание поездки и связанного доступа |
| **Цель** | Проверить, что метод createTrip успешно создает сущность Trip, устанавливает создателя, сохраняет ее через tripRepository, создает и сохраняет запись TripAccess для создателя с правами 'admin' и возвращает DTO созданной поездки. |
| **Функция** | TripServiceImpl.createTrip |
| **Предусловия** | Моки UserService (возвращает currentUser по ID), TripMapper (преобразует DTO в сущность и обратно), TripRepository (возвращает сохраненную сущность savedTripEntity), TripAccessRepository (возвращает сохраненную сущность TripAccess). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | currentUserId (Long), tripDtoToCreate (TripDto). |
| **Шаги выполнения** | 1. Настроить моки для userService.getUserEntityById, tripMapper.toEntity, tripRepository.save, tripAccessRepository.save, tripMapper.toDto.   2. Вызвать tripService.createTrip(currentUserId, tripDtoToCreate).   3. Перехватить результат (TripDto). |
| **Ожидаемый результат** | Метод возвращает TripDto, соответствующий createdTripDto. Сущность Trip сохраняется с установленным creator и deleted=false. Запись TripAccess сохраняется для создателя с уровнем доступа 'admin'. |
| **Проверка** | Проверка, что результат не null и поля соответствуют createdTripDto. Проверка вызовов userService.getUserEntityById, tripMapper.toEntity, tripRepository.save, tripAccessRepository.save, tripMapper.toDto с корректными аргументами. |
| **Полученный результат** | Функция успешно создала поездку и запись доступа, вернула корректное DTO. Вызовы зависимостей корректны. |

* + 1. Создание поездки (пользователь не найден) (createTrip\_UserNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Создание поездки (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с currentUserId не найден (userService.getUserEntityById выбрасывает исключение), метод createTrip пробрасывает ResourceNotFoundException. |
| **Функция** | TripServiceImpl.createTrip |
| **Предусловия** | Мок UserService. userService.getUserEntityById настроен на выброс ResourceNotFoundException. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | nonExistentUserId (Long), tripDtoToCreate (TripDto). |
| **Шаги выполнения** | 1. Настроить userService.getUserEntityById на выброс исключения.   2. Вызвать tripService.createTrip(nonExistentUserId, tripDtoToCreate) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException. |
| **Проверка** | Проверка типа выброшенного исключения. Убедиться, что методы tripMapper.toEntity, tripRepository.save, tripAccessRepository.save не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. Создание поездки не произошло. |

* + 1. Создание поездки (ошибка репозитория) (createTrip\_RepositoryError)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Создание поездки (ошибка при сохранении в репозитории) |
| **Цель** | Проверить, что если tripRepository.save выбрасывает исключение, метод createTrip пробрасывает это исключение дальше. |
| **Функция** | TripServiceImpl.createTrip |
| **Предусловия** | Моки UserService (возвращает пользователя), TripMapper (преобразует DTO в сущность). Мок TripRepository. tripRepository.save настроен на выброс RuntimeException. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | currentUserId (Long), tripDtoToCreate (TripDto). |
| **Шаги выполнения** | 1. Настроить моки до tripRepository.save.   2. Настроить tripRepository.save на выброс исключения.   3. Вызвать tripService.createTrip(currentUserId, tripDtoToCreate) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается RuntimeException. |
| **Проверка** | Проверка типа выброшенного исключения. Проверка вызовов userService.getUserEntityById, tripMapper.toEntity, tripRepository.save. Убедиться, что tripAccessRepository.save и tripMapper.toDto не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение RuntimeException. Доступ не создавался, DTO не возвращалось. |

* + 1. Успешное получение поездки по ID (getTripById\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное получение поездки по ID |
| **Цель** | Проверить, что метод getTripById успешно находит поездку, проверяет доступ пользователя (в данном случае, т.к. пользователь - создатель, доступ есть) и возвращает DTO поездки. |
| **Функция** | TripServiceImpl.getTripById (включая внутренний вызов getTripEntityWithAccessCheck) |
| **Предусловия** | Моки UserService (возвращает пользователя), TripRepository (возвращает Optional.of(savedTripEntity)), TripMapper (преобразует сущность в DTO). savedTripEntity.creator совпадает с currentUser. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), tripId (Long). |
| **Шаги выполнения** | 1. Настроить моки userService.getUserEntityById, tripRepository.findById, tripMapper.toDto.   2. Вызвать tripService.getTripById(userId, tripId).   3. Перехватить результат (TripDto). |
| **Ожидаемый результат** | Метод возвращает TripDto, соответствующий createdTripDto. |
| **Проверка** | Проверка, что результат не null и поля соответствуют createdTripDto. Проверка вызовов userService.getUserEntityById (дважды), tripRepository.findById, tripMapper.toDto. |
| **Полученный результат** | Функция успешно вернула DTO найденной поездки. Вызовы зависимостей корректны. |

* + 1. Получение поездки по ID (поездка не найдена) (getTripById\_TripNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение поездки по ID (поездка не найдена) |
| **Цель** | Проверить, что если поездка с tripId не найдена (tripRepository.findById возвращает empty), метод getTripById выбрасывает ResourceNotFoundException. |
| **Функция** | TripServiceImpl.getTripById (включая внутренний вызов getTripEntityWithAccessCheck) |
| **Предусловия** | Мок UserService (возвращает пользователя). Мок TripRepository. tripRepository.findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | userId (Long), nonExistentTripId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.getTripById(userId, nonExistentTripId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException. |
| **Проверка** | Проверка типа выброшенного исключения. Проверка вызовов userService.getUserEntityById, tripRepository.findById. Убедиться, что tripMapper.toDto не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. |

* + 1. Получение поездки по ID (доступ запрещен) (getTripById\_AccessDenied)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение поездки по ID (доступ запрещен) |
| **Цель** | Проверить, что если пользователь не является создателем поездки и не имеет записи в TripAccess, метод getTripById выбрасывает AccessDeniedException. |
| **Функция** | TripServiceImpl.getTripById (включая внутренний вызов getTripEntityWithAccessCheck) |
| **Предусловия** | Моки UserService (возвращает anotherUser), TripRepository (возвращает savedTripEntity, созданный currentUser), TripAccessRepository (возвращает Optional.empty() для anotherUser). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | anotherUserId (Long), tripId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.getTripById(anotherUserId, tripId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается AccessDeniedException с сообщением "У вас нет доступа к этой поездке". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызовов userService.getUserEntityById, tripRepository.findById, tripAccessRepository.findByTripAndUser. Убедиться, что tripMapper.toDto не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение AccessDeniedException. |

* + 1. Успешное обновление поездки (updateTrip\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное обновление поездки |
| **Цель** | Проверить, что метод updateTrip успешно находит поездку, проверяет доступ (пользователь - создатель), обновляет поля сущности через маппер, сохраняет изменения и возвращает обновленное DTO. |
| **Функция** | TripServiceImpl.updateTrip |
| **Предусловия** | Моки UserService (возвращает пользователя), TripRepository (находит поездку, save возвращает обновленную сущность), TripAccessRepository (для проверки доступа, если нужно), TripMapper (обновляет сущность из DTO updateEntityFromDto, преобразует сущность в DTO toDto). Пользователь - создатель. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), tripId (Long), tripDtoToUpdate (TripDto). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.updateTrip(userId, tripId, tripDtoToUpdate).   3. Перехватить результат (TripDto) и захватить аргумент для tripMapper.updateEntityFromDto. |
| **Ожидаемый результат** | Метод возвращает TripDto с обновленными данными (updatedTripDto). Вызывается tripMapper.updateEntityFromDto для обновления сущности. Обновленная сущность сохраняется. |
| **Проверка** | Проверка, что результат не null и содержит обновленные поля. Проверка вызовов userService.getUserEntityById (дважды), tripRepository.findById, tripMapper.updateEntityFromDto, tripRepository.save, tripMapper.toDto. Проверка, что в updateEntityFromDto передана найденная сущность. |
| **Полученный результат** | Функция успешно обновила поездку и вернула DTO. Вызовы зависимостей корректны. |

* + 1. Обновление поездки (поездка не найдена) (updateTrip\_TripNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Обновление поездки (поездка не найдена) |
| **Цель** | Проверить, что если поездка с tripId не найдена при проверке доступа, метод updateTrip выбрасывает ResourceNotFoundException. |
| **Функция** | TripServiceImpl.updateTrip |
| **Предусловия** | Мок UserService (возвращает пользователя). Мок TripRepository. tripRepository.findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | userId (Long), nonExistentTripId (Long), tripDtoToUpdate (TripDto). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.updateTrip(userId, nonExistentTripId, tripDtoToUpdate) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException. |
| **Проверка** | Проверка типа выброшенного исключения. Проверка вызовов userService.getUserEntityById, tripRepository.findById. Убедиться, что tripMapper и tripRepository.save не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. Обновление не произошло. |

* + 1. Обновление поездки (доступ запрещен) (updateTrip\_AccessDenied)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Обновление поездки (доступ запрещен) |
| **Цель** | Проверить, что если пользователь не имеет прав на редактирование поездки (не создатель и нет записи 'admin'/'write' в TripAccess), метод updateTrip выбрасывает AccessDeniedException. |
| **Функция** | TripServiceImpl.updateTrip |
| **Предусловия** | Моки UserService (возвращает anotherUser), TripRepository (возвращает savedTripEntity), TripAccessRepository (возвращает Optional.empty() или запись с 'read' доступом). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | anotherUserId (Long), tripId (Long), tripDtoToUpdate (TripDto). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.updateTrip(anotherUserId, tripId, tripDtoToUpdate) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается AccessDeniedException с сообщением "У вас нет доступа к этой поездке". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызовов userService.getUserEntityById, tripRepository.findById, tripAccessRepository.findByTripAndUser. Убедиться, что tripMapper и tripRepository.save не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение AccessDeniedException. Обновление не произошло. |

* + 1. Успешное (мягкое) удаление поездки (deleteTrip\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Успешное мягкое удаление поездки |
| **Цель** | Проверить, что метод deleteTrip успешно находит поездку, проверяет доступ (пользователь - создатель), устанавливает флаг deleted в true и сохраняет изменения. |
| **Функция** | TripServiceImpl.deleteTrip |
| **Предусловия** | Моки UserService (возвращает пользователя), TripRepository (находит поездку, save мокирован для захвата аргумента). Пользователь - создатель. |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), tripId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.deleteTrip(userId, tripId).   3. Захватить аргумент для tripRepository.save. |
| **Ожидаемый результат** | Метод успешно завершается. Сущность Trip, переданная в tripRepository.save, имеет deleted=true. |
| **Проверка** | Проверка вызовов userService.getUserEntityById (дважды), tripRepository.findById, tripRepository.save. С помощью tripCaptor проверить, что у сохраненной сущности isDeleted() возвращает true. |
| **Полученный результат** | Функция успешно выполнила мягкое удаление поездки. Вызовы зависимостей корректны. |

* + 1. Удаление поездки (поездка не найдена) (deleteTrip\_TripNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Удаление поездки (поездка не найдена) |
| **Цель** | Проверить, что если поездка с tripId не найдена при проверке доступа, метод deleteTrip выбрасывает ResourceNotFoundException. |
| **Функция** | TripServiceImpl.deleteTrip |
| **Предусловия** | Мок UserService (возвращает пользователя). Мок TripRepository. tripRepository.findById возвращает Optional.empty(). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | userId (Long), nonExistentTripId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.deleteTrip(userId, nonExistentTripId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException. |
| **Проверка** | Проверка типа выброшенного исключения. Проверка вызовов userService.getUserEntityById, tripRepository.findById. Убедиться, что tripRepository.save не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. Удаление не произошло. |

* + 1. Удаление поездки (доступ запрещен) (deleteTrip\_AccessDenied)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Удаление поездки (доступ запрещен) |
| **Цель** | Проверить, что если пользователь не имеет прав администратора (не создатель и нет 'admin' в TripAccess), метод deleteTrip выбрасывает AccessDeniedException. |
| **Функция** | TripServiceImpl.deleteTrip |
| **Предусловия** | Моки UserService (возвращает anotherUser), TripRepository (возвращает savedTripEntity), TripAccessRepository (возвращает Optional.empty() или запись не с 'admin'). |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | anotherUserId (Long), tripId (Long). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.deleteTrip(anotherUserId, tripId) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается AccessDeniedException с сообщением "У вас нет доступа к этой поездке". |
| **Проверка** | Проверка типа и сообщения исключения. Проверка вызовов userService.getUserEntityById, tripRepository.findById, tripAccessRepository.findByTripAndUser. Убедиться, что tripRepository.save не вызывался. |
| **Полученный результат** | Функция выбросила ожидаемое исключение AccessDeniedException. Удаление не произошло. |

* + 1. Получение созданных поездок пользователя (getUserTrips\_FilterCreated\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение созданных поездок пользователя |
| **Цель** | Проверить, что метод getUserTrips с фильтром 'created' вызывает tripRepository.findAllByCreator и возвращает страницу DTO поездок. |
| **Функция** | TripServiceImpl.getUserTrips |
| **Предусловия** | Моки UserService (возвращает currentUser), TripRepository (метод findAllByCreator возвращает Page<Trip>), TripMapper (преобразует сущности в DTO). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), filter="created", pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.getUserTrips(userId, "created", pageable).   3. Перехватить результат (Page<TripDto>). |
| **Ожидаемый результат** | Метод возвращает Page<TripDto> с поездками, созданными пользователем. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызовов userService.getUserEntityById, tripRepository.findAllByCreator, tripMapper.toDto. Убедиться, что другие методы репозитория (findAllSharedWithUser, findAllAvailableToUser) не вызывались. |
| **Полученный результат** | Функция успешно вернула страницу созданных поездок. Вызовы зависимостей корректны. |

* + 1. Получение расшаренных поездок пользователя (getUserTrips\_FilterShared\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение расшаренных поездок пользователя |
| **Цель** | Проверить, что метод getUserTrips с фильтром 'shared' вызывает tripRepository.findAllSharedWithUser и возвращает страницу DTO поездок. |
| **Функция** | TripServiceImpl.getUserTrips |
| **Предусловия** | Моки UserService (возвращает currentUser), TripRepository (метод findAllSharedWithUser возвращает Page<Trip>), TripMapper (преобразует сущности в DTO). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), filter="shared", pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.getUserTrips(userId, "shared", pageable).   3. Перехватить результат (Page<TripDto>). |
| **Ожидаемый результат** | Метод возвращает Page<TripDto> с поездками, к которым пользователю предоставили доступ. |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызовов userService.getUserEntityById, tripRepository.findAllSharedWithUser, tripMapper.toDto. Убедиться, что другие методы репозитория (findAllByCreator, findAllAvailableToUser) не вызывались. |
| **Полученный результат** | Функция успешно вернула страницу расшаренных поездок. Вызовы зависимостей корректны. |

* + 1. Получение всех доступных поездок пользователя (getUserTrips\_FilterAll\_Success)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение всех доступных поездок пользователя |
| **Цель** | Проверить, что метод getUserTrips с фильтром 'all' вызывает tripRepository.findAllAvailableToUser и возвращает страницу DTO поездок. |
| **Функция** | TripServiceImpl.getUserTrips |
| **Предусловия** | Моки UserService (возвращает currentUser), TripRepository (метод findAllAvailableToUser возвращает Page<Trip>), TripMapper (преобразует сущности в DTO). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), filter="all", pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.getUserTrips(userId, "all", pageable).   3. Перехватить результат (Page<TripDto>). |
| **Ожидаемый результат** | Метод возвращает Page<TripDto> со всеми поездками, доступными пользователю (созданные + расшаренные). |
| **Проверка** | Проверка, что результат не null и соответствует данным из моков. Проверка вызовов userService.getUserEntityById, tripRepository.findAllAvailableToUser, tripMapper.toDto. Убедиться, что другие методы репозитория (findAllByCreator, findAllSharedWithUser) не вызывались. |
| **Полученный результат** | Функция успешно вернула страницу всех доступных поездок. Вызовы зависимостей корректны. |

* + 1. Получение поездок пользователя (нет результатов) (getUserTrips\_NoResults)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение поездок пользователя (нет результатов) |
| **Цель** | Проверить, что если соответствующий метод репозитория возвращает пустую страницу (Page.empty), метод getUserTrips также возвращает пустую страницу. |
| **Функция** | TripServiceImpl.getUserTrips |
| **Предусловия** | Мок UserService (возвращает пользователя). Мок TripRepository. Метод findAllByCreator (или другой, в зависимости от фильтра) возвращает Page.empty(). |
| **Тип тестирования** | Позитивный |
| **Данные запроса** | userId (Long), filter="created", pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить моки.   2. Вызвать tripService.getUserTrips(userId, "created", pageable).   3. Перехватить результат (Page<TripDto>). |
| **Ожидаемый результат** | Метод возвращает пустую Page<TripDto>. |
| **Проверка** | Проверка, что результат isEmpty() возвращает true и getTotalElements() возвращает 0. Проверка вызовов userService.getUserEntityById, соответствующего метода TripRepository. Убедиться, что tripMapper.toDto не вызывался. |
| **Полученный результат** | Функция успешно вернула пустую страницу, как и ожидалось. Вызовы зависимостей корректны. |

* + 1. Получение поездок пользователя (пользователь не найден) (getUserTrips\_UserNotFound)

|  |  |
| --- | --- |
| **Параметр** | **Описание** |
| **Название теста** | Unit-тест: Получение поездок пользователя (пользователь не найден) |
| **Цель** | Проверить, что если пользователь с userId не найден (userService.getUserEntityById выбрасывает исключение), метод getUserTrips пробрасывает ResourceNotFoundException. |
| **Функция** | TripServiceImpl.getUserTrips |
| **Предусловия** | Мок UserService. userService.getUserEntityById настроен на выброс ResourceNotFoundException. |
| **Тип тестирования** | Негативный (проверка обработки ошибки) |
| **Данные запроса** | nonExistentUserId (Long), filter (String), pageable (Pageable). |
| **Шаги выполнения** | 1. Настроить userService.getUserEntityById на выброс исключения.   2. Вызвать tripService.getUserTrips(nonExistentUserId, filter, pageable) внутри assertThrows. |
| **Ожидаемый результат** | Выбрасывается ResourceNotFoundException. |
| **Проверка** | Проверка типа выброшенного исключения. Убедиться, что методы TripRepository и TripMapper не вызывались. |
| **Полученный результат** | Функция выбросила ожидаемое исключение ResourceNotFoundException. Поиск поездок не производился. |

1. Выявленные дефекты

Дефектов не обнаружено в ходе тестирования основных функций.

1. Общая оценка

• •Юнит-тесты проходят успешно

• Логика API работает корректно при изолированном тестировании

• Покрытие основных бизнес-сценариев достигнуто