Документация тестирования

Тестирование Backend

Выполнил: Вологжин Егор Владимирович, Тестировщик

1 Тест: register_WhenEmailIsFree_ShouldReturnOk

Описание:

Этот тест проверяет, что при попытке регистрации с использованием свободного email (когда в системе нет пользователя с таким email), контроллер возвращает успешный ответ с кодом 200 OK и сообщением "The mail is free".

Область тестирования:

— Проверка механизма регистрации пользователя, а именно — обработка случая, когда указанный email свободен.

— Взаимодействие с сервисом UserService для проверки наличия пользователя по email.

Шаги выполнения теста:

1. Создается объект UserCheckEmailDTO, в который передается email free@mail.com.
2. Мокируется поведение метода userService.getUserByEmail(email), чтобы он возвращал null (что означает, что пользователя с таким email нет в системе).
3. Вызывается метод register(dto) контроллера authController, который должен проверить, свободен ли email, и если он свободен, вернуть соответствующий ответ.
4. Проверяется, что возвращается правильный HTTP статус (200 OK).

5. Проверяется, что в теле ответа содержится строка "The mail is free".
6. Проверяется, что метод getUserByEmail(email) был вызван с правильным параметром email.

Примерный вывод из теста:

Если email свободен, то система сообщает об этом и разрешает регистрацию.

Вывод:

Этот тест удостоверяется, что логика проверки свободности email работает правильно, и что система возвращает правильное сообщение и статус.

2 Тест: register_WhenEmailIsBusy_ShouldReturnConflict

Описание:

Этот тест проверяет, что при попытке регистрации с использованием уже занятого email (когда в системе есть пользователь с таким email), контроллер возвращает статус 409 Conflict с сообщением "Mail is busy".

Область тестирования:

— Проверка механизма регистрации пользователя, а именно — обработка случая, когда указанный email уже занят.

— Взаимодействие с сервисом UserService для проверки наличия пользователя по email.

Шаги выполнения теста:

1. Создается объект UserCheckEmailDTO, в который передается email test@example.com.
2. Создается объект User, который имитирует уже существующего пользователя с указанным email.
3. Мокируется поведение метода userService.getUserByEmail(email), чтобы он возвращал созданного пользователя с этим email (тем самым имитируя ситуацию, когда email занят).
4. Вызывается метод register(dto) контроллера authController, который должен проверить, занят ли email, и если он занят, вернуть ответ с кодом 409 Conflict.
5. Проверяется, что возвращается правильный HTTP статус — 409 Conflict.

6. Проверяется, что в теле ответа содержится сообщение "Mail is busy".
7. Проверяется, что метод userService.getUserByEmail(email) был вызван с правильным параметром email.

Примерный вывод из теста:

Если email уже занят, система должна вернуть статус 409 Conflict и сообщить, что такой email уже используется, чтобы не было возможности зарегистрировать нового пользователя с этим же email.

Вывод:

Этот тест удостоверяется, что логика проверки свободности email работает правильно, и что система возвращает правильное сообщение и статус.

3 Тест: registerMedicalCard_ShouldReturnCreated

Описание:

Этот тест проверяет, что при регистрации пользователя с медицинской картой система возвращает статус 201 Created и сообщение "User and Medical Card created successfully", а также что метод для регистрации пользователя с медицинской картой был вызван.

Область тестирования:

— Регистрация пользователя с медицинской картой через контроллер.

— Взаимодействие с сервисом UserService для создания пользователя с медицинской картой.

Шаги выполнения теста:

1. Создается объект MedicalCardDTO, который представляет данные для регистрации медицинской карты.
2. Вызывается метод registerMedicalCard(dto) контроллера authController, который должен зарегистрировать пользователя вместе с медицинской картой.
3. Проверяется, что возвращается статус 201 Created, который означает успешное создание ресурса (в данном случае — пользователя и медицинской карты).
4. Проверяется, что в теле ответа содержится сообщение "User and Medical Card created successfully".
5. Проверяется, что метод userService.registerUserWithMedicalCard(dto) был вызван с переданным объектом dto.

Примерный вывод из теста:

Если регистрация с медицинской картой прошла успешно, то система должна вернуть статус 201 Created и соответствующее сообщение. Метод для регистрации пользователя с медицинской картой должен быть вызван без ошибок.

Вывод:

Этот тест удостоверяется, что при регистрации пользователя с медицинской картой система корректно обрабатывает запрос и возвращает правильный статус и сообщение, а также что все необходимые сервисные методы вызываются.

4 Тест: login_ShouldReturnJwtTokens

Описание:

Этот тест проверяет, что при успешной аутентификации пользователя с использованием правильных учетных данных система возвращает JWT токены (access и refresh), а также что сервис аутентификации был правильно вызван.

Область тестирования:

— Аутентификация пользователя.

— Генерация JWT токенов (access и refresh).

— Взаимодействие с сервисом UserService для аутентификации и получения токенов.

Шаги выполнения теста:

1. Создается объект UserLoginDTO, в который передаются email и пароль пользователя (test@example.com и password123).
2. Создается объект JwtResponse, который представляет ожидаемые JWT токены (access и refresh) для успешной аутентификации.
3. Мокируется поведение метода userService.authenticate(dto), чтобы он возвращал объект JwtResponse с заранее заданными токенами.
4. Вызывается метод login(dto) контроллера authController, который должен проверить учетные данные и вернуть соответствующие JWT токены.

5. Проверяется, что статус ответа равен 200 OK, что подтверждает успешную аутентификацию.
6. Проверяется, что в теле ответа содержится объект JwtResponse с правильными токенами.
7. Проверяется, что метод userService.authenticate(dto) был вызван с переданным объектом dto.

Примерный вывод из теста:

Если учетные данные верны, система должна вернуть статус 200 OK и соответствующие JWT токены (access и refresh). Метод аутентификации должен быть вызван с переданными параметрами и возвращать правильные токены.

Вывод:

Этот тест удостоверяется, что при правильных учетных данных система корректно выполняет аутентификацию, возвращает JWT токены и взаимодействует с соответствующим сервисом для их генерации.

5 Тест: Тест: sendRecoveryCode_ShouldReturnOk

Описание:

Этот тест проверяет, что при запросе кода для восстановления пароля на указанный email, система успешно генерирует и отправляет код восстановления, а затем возвращает статус 200 OK с сообщением "Recovery code sent to email". Также проверяется, что соответствующие методы сервиса для генерации и отправки кода были вызваны.

Область тестирования:

— Восстановление пароля через отправку кода на email.

— Взаимодействие с сервисом PasswordRecoveryService для генерации и отправки кода.

Шаги выполнения теста:

1. Создается объект EmailRequest, в который передается email пользователя (test@example.com), на который будет отправлен код для восстановления пароля.
2. Мокируется поведение метода passwordRecoveryService.generateCode(), чтобы он возвращал заранее заданный код ("123456").
3. Мокируется поведение метода passwordRecoveryService.sendRecoveryCode(email, generatedCode), чтобы при его вызове не происходило ошибок (с помощью doNothing()).
4. Вызывается метод sendRecoveryCode(request) контроллера authController, который должен инициировать процесс отправки кода на email.

5. Проверяется, что возвращается статус 200 OK, подтверждающий успешную операцию.
6. Проверяется, что в теле ответа содержится сообщение "Recovery code sent to email".
7. Проверяется, что метод passwordRecoveryService.generateCode() был вызван для генерации кода.
8. Проверяется, что метод passwordRecoveryService.sendRecoveryCode(email, generatedCode) был вызван для отправки кода на email.

Примерный вывод из теста:

Если запрос на восстановление пароля прошел успешно, система должна вернуть статус 200 OK и сообщение о том, что код восстановления был отправлен на email. Методы генерации и отправки кода должны быть вызваны корректно.

Вывод:

Этот тест удостоверяется, что процесс восстановления пароля работает корректно при успешной генерации и отправке кода, а также что соответствующие методы сервиса были вызваны.

6 Тест: sendRecoveryCode_WhenMessagingException_ShouldReturnServerError

Описание:

Этот тест проверяет, что если при попытке отправки кода для восстановления пароля возникает ошибка (например, проблема с почтовым сервером), система возвращает статус 500 Internal Server Error и сообщение "Failed to send email".

Область тестирования:

— Обработка ошибок при отправке email для восстановления пароля.

— Проверка правильности обработки исключений и возврата соответствующего статуса ошибки.

Шаги выполнения теста:

1. Создается объект EmailRequest, в который передается email пользователя (test@example.com), на который будет отправлен код для восстановления пароля.
2. Мокируется поведение метода passwordRecoveryService.generateCode() для возврата заранее заданного кода ("123456").
3. Мокируется поведение метода passwordRecoveryService.sendRecoveryCode(email, generatedCode), чтобы он выбросил исключение MessagingException (симулируя ошибку при отправке email).

4. Вызывается метод sendRecoveryCode(emailRequest) контроллера authController, который должен обработать ошибку и вернуть корректный ответ.
5. Проверяется, что возвращается статус 500 Internal Server Error, что подтверждает наличие ошибки на сервере.
6. Проверяется, что в теле ответа содержится сообщение "Failed to send email", что сигнализирует о сбое при отправке email.

Примерный вывод из теста:

Если при отправке email возникает ошибка (например, проблема с почтовым сервером), система должна вернуть статус 500 Internal Server Error с соответствующим сообщением.

Важно, чтобы система правильно обрабатывала исключение и не продолжала выполнение в случае ошибки.

Вывод:

Этот тест удостоверяется, что система корректно обрабатывает ошибки, связанные с отправкой email, и возвращает соответствующий статус и сообщение об ошибке в случае сбоя.

7 Тест: checkRecoveryCode_WhenValid_ShouldReturnOk

Описание:

Этот тест проверяет, что при правильном коде для восстановления пароля, система возвращает статус 200 OK и сообщение "Code verified successfully".

Область тестирования:

— Проверка механизма верификации кода для восстановления пароля.

— Взаимодействие с сервисом PasswordRecoveryService для верификации кода.

Шаги выполнения теста:

1. Создается объект CodeVerificationRequest, в который передаются email пользователя (test@example.com) и код восстановления пароля (123456).
2. Мокируется поведение метода passwordRecoveryService.verifyCode(email, String.valueOf(code)), чтобы он возвращал true, что означает, что код верный.
3. Вызывается метод checkRecoveryCode(codeVerificationRequest) контроллера authController, который должен проверить правильность введенного кода и вернуть успешный ответ.
4. Проверяется, что возвращается статус 200 OK, подтверждающий успешную верификацию кода.
5. Проверяется, что в теле ответа содержится сообщение "Code verified successfully".

Примерный вывод из теста:

Если введенный код для восстановления пароля правильный, система должна вернуть статус 200 OK и сообщение о том, что код был успешно верифицирован.

Вывод:

Этот тест удостоверяется, что система корректно проверяет введенный код для восстановления пароля и возвращает правильный ответ в случае успеха.

8 Тест: checkRecoveryCode_WhenInvalid_ShouldReturnBadRequest

Описание:

Этот тест проверяет, что при введении неправильного кода для восстановления пароля система возвращает статус 400 Bad Request с сообщением "Invalid code".

Область тестирования:

— Проверка механизма верификации кода для восстановления пароля.

— Взаимодействие с сервисом PasswordRecoveryService для верификации кода.

Шаги выполнения теста:

1. Создается объект CodeVerificationRequest, в который передаются email пользователя (test@example.com) и код восстановления пароля (123456).
2. Мокируется поведение метода passwordRecoveryService.verifyCode(email, String.valueOf(code)), чтобы он возвращал false, что означает, что код неправильный.
3. Вызывается метод checkRecoveryCode(codeVerificationRequest) контроллера authController, который должен проверить неправильность введенного кода и вернуть ошибку.
4. Проверяется, что возвращается статус 400 Bad Request, что подтверждает, что код некорректен.

5. Проверяется, что в теле ответа содержится сообщение "Invalid code", информируя пользователя о неправильном коде.

Примерный вывод из теста:

Если введенный код для восстановления пароля неправильный, система должна вернуть статус 400 Bad Request с сообщением "Invalid code".

Вывод:

Этот тест удостоверяется, что система корректно реагирует на введение неверного кода для восстановления пароля и возвращает правильный статус и сообщение об ошибке.

9 Тест: resetPassword_ShouldReturnOk

Описание:

Этот тест проверяет, что при успешном сбросе пароля система возвращает статус 200 OK с сообщением "Password reset successfully", а также что сервис для сброса пароля был корректно вызван.

Область тестирования:

— Сброс пароля пользователя. Взаимодействие с сервисом

— PasswordRecoveryService для сброса пароля.

Шаги выполнения теста:

1. Создается объект PasswordResetRequest, в который передаются email пользователя (test@example.com) и новый пароль (newPassword123).
2. Мокируется поведение метода passwordRecoveryService.resetPassword(email, newPassword), чтобы не возникало ошибок при вызове (с помощью doNothing()).
3. Вызывается метод resetPassword(passwordResetRequest) контроллера authController, который должен инициировать процесс сброса пароля.
4. Проверяется, что возвращается статус 200 OK, подтверждающий успешное выполнение операции.
5. Проверяется, что в теле ответа содержится сообщение "Password reset successfully".

6. Проверяется, что метод passwordRecoveryService.resetPassword(email, newPassword) был вызван с переданными значениями.

Примерный вывод из теста:

Если сброс пароля прошел успешно, система должна вернуть статус 200 OK и сообщение о том, что пароль был успешно сброшен. Метод для сброса пароля должен быть вызван с правильными параметрами.

Вывод:

Этот тест удостоверяется, что при сбросе пароля система корректно выполняет операцию, возвращая правильный статус и сообщение, а также что метод сброса пароля в сервисе был вызван с нужными параметрами.

10 Тест: getMedicalCard_ShouldReturnOk

Описание:

Этот тест проверяет, что при запросе медицинской карты пользователя система возвращает статус 200 OK и ожидаемый объект с данными медицинской карты.

Область тестирования:

— Получение данных о медицинской карте пользователя.

— Взаимодействие с сервисом MedicalCardService для получения данных.

Шаги выполнения теста:

1. Создается объект MedicalCardResponse, в который устанавливаются данные медицинской карты (например, полное имя, рост, вес, группа крови, аллергии, заболевания).
2. Мокируется поведение метода medicalCardService.getMedicalCard(), чтобы он возвращал заранее подготовленный объект expectedResponse.
3. Вызывается метод getMedicalCard() контроллера medicalCardController, который должен вернуть данные медицинской карты.
4. Проверяется, что возвращается статус 200 OK, подтверждающий успешный запрос.
5. Проверяется, что в теле ответа содержится объект expectedResponse, который должен быть равен возвращаемому объекту.

6. Проверяется, что ме-
   тод medicalCardService.getMedicalCard() был вызван.

Примерный вывод из теста:

Если запрос медицинской карты прошел успешно, система должна вернуть статус 200 OK и правильный объект с данными медицинской карты. Метод получения данных о медицинской карте должен быть вызван и вернуть корректные данные.

Вывод:

Этот тест удостоверяется, что при запросе медицинской карты система корректно выполняет операцию получения данных и возвращает правильный статус и объект с данными.

11 Тест: editMedicalCard_ShouldReturnOk

Описание:

Этот тест проверяет, что при обновлении данных медицинской карты пользователя система возвращает статус 200 OK и обновленные данные медицинской карты.

Область тестирования:

— Обновление данных медицинской карты пользователя.

— Взаимодействие с сервисом MedicalCardService для редактирования медицинской карты.

Шаги выполнения теста:

1. Создается объект MedicalCardRequest, в который устанавливаются новые данные для обновления медицинской карты (например, имя, рост, вес, группа крови, аллергии, заболевания).
2. Создается объект MedicalCardResponse, который содержит обновленные данные медицинской карты, которые будут возвращены после успешного редактирования.
3. Мокируется поведение метода medicalCardService.editMedicalCard(request), чтобы он возвращал объект responseDto (обновленные данные).
4. Вызывается метод editMedicalCard(request) контроллера medicalCardController, который должен обновить данные и вернуть результат.
5. Проверяется, что возвращается статус 200 OK, подтверждающий успешное обновление данных.

6. Проверяется, что в теле ответа содержится объект responseDto, который должен быть равен возвращаемому объекту.
7. Проверяется, что метод medicalCardService.editMedicalCard(request) был вызван с переданным объектом request.

Примерный вывод из теста:

Если данные медицинской карты были успешно обновлены, система должна вернуть статус 200 OK и объект с обновленными данными. Метод редактирования медицинской карты должен быть вызван и обработать запрос правильно.

Вывод:

Этот тест удостоверяется, что при обновлении данных медицинской карты система корректно выполняет операцию, возвращая правильный статус и обновленные данные.

12 Тест: getMedicationSchedule_ShouldReturnListOfSchedules

Описание:

Этот тест проверяет, что при запросе списка расписаний приема медикаментов система возвращает статус 200 OK и корректный список расписаний.

Область тестирования:

— Получение списка расписаний медикаментов для пользователя.

— Взаимодействие с сервисом MedicationScheduleService для получения расписаний.

Шаги выполнения теста:

1. Создаются два объекта MedicationSchedule (например, firstSchedule и secondSchedule), которые будут представлять два расписания.
2. Создается список schedules, который содержит эти два расписания.
3. Мокируется поведение метода medicationScheduleService.getUserSchedules(), чтобы он возвращал заранее подготовленный список расписаний.
4. Вызывается метод getUserSchedules() контроллера medicationScheduleController, который должен вернуть список расписаний.
5. Проверяется, что возвращается статус 200 OK, подтверждающий успешный запрос.

6. Проверяется, что в теле ответа содержится список расписаний, который должен быть равен заранее подготовленному списку schedules.
7. Проверяется, что метод medicationScheduleService.getUserSchedules() был вызван.

Примерный вывод из теста:

Если запрос на получение расписаний прошел успешно, система должна вернуть статус 200 OK и список расписаний. Метод получения расписаний должен быть вызван и вернуть корректные данные.

Вывод:

Этот тест удостоверяется, что при запросе расписаний система корректно выполняет операцию получения списка данных и возвращает правильный статус и объект.

13 Тест: addSchedule_ShouldReturnCreatedSchedule

Описание:

Этот тест проверяет, что при добавлении нового расписания для приема медикамента система возвращает статус 201 Created и корректно созданное расписание.

Область тестирования:

— Добавление нового расписания для приема медикаментов.

— Взаимодействие с сервисом MedicationScheduleService для добавления расписания.

Шаги выполнения теста:

1. Создается объект MedicationScheduleRequest, в который устанавливаются данные для нового расписания (например, название лекарства "Ibuprofen", дни (1, 3, 5) и время приема (8:00 и 20:00)).
2. Создается объект MedicationSchedule, который представляет собой расписание, которое будет возвращено после его успешного создания.
3. Мокируется поведение метода medicationScheduleService.addSchedule(request), чтобы он возвращал заранее подготовленное расписание createsSchedule.
4. Вызывается метод addSchedule(request) контроллера medicationScheduleController, который должен создать новое расписание и вернуть его.
5. Проверяется, что возвращается статус 201 Created, подтверждающий успешное создание ресурса.

6. Проверяется, что в теле ответа содержится объект createsSchedule, который должен быть равен создаваемому расписанию.
7. Проверяется, что метод medicationScheduleService.addSchedule(request) был вызван с переданным объектом request.

Примерный вывод из теста:

Если новое расписание было успешно добавлено, система должна вернуть статус 201 Created и объект с данными созданного расписания. Метод добавления расписания должен быть вызван и обработать запрос корректно.

Вывод:

Этот тест удостоверяется, что при добавлении нового расписания система корректно выполняет операцию создания, возвращая правильный статус и объект.

14 Тест: editMedicationSchedule_ShouldReturnUpdatedSchedule

Описание:

Этот тест проверяет, что при редактировании существующего расписания приема медикаментов система возвращает статус 200 OK и обновленное расписание.

Область тестирования:

— Редактирование существующего расписания приема медикаментов.

— Взаимодействие с сервисом MedicationScheduleService для обновления расписания.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для расписания, которое будет редактироваться.
2. Создается объект MedicationScheduleRequest, в который устанавливаются новые данные для редактируемого расписания (например, название лекарства "Paracetamol", дни (2, 4, 6) и время приема (9:00 и 21:00)).
3. Создается объект MedicationSchedule, который представляет обновленное расписание, которое будет возвращено после успешного редактирования.
4. Мокируется поведение метода medicationScheduleService.editSchedule(id, request), чтобы он возвращал заранее подготовленное обновленное расписание updatedSchedule.

5. Вызывается метод editSchedule(id, request) контроллера medicationScheduleController, который должен обновить расписание и вернуть результат.
6. Проверяется, что возвращается статус 200 OK, подтверждающий успешное редактирование ресурса.
7. Проверяется, что в теле ответа содержится объект updatedSchedule, который должен быть равен возвращаемому обновленному расписанию.
8. Проверяется, что метод medicationScheduleService.editSchedule(id, request) был вызван с правильными параметрами (id и request).

Примерный вывод из теста:

Если расписание было успешно обновлено, система должна вернуть статус 200 OK и объект с обновленными данными. Метод редактирования расписания должен быть вызван и обработать запрос корректно.

Вывод:

Этот тест удостоверяется, что при редактировании расписания система корректно выполняет операцию обновления, возвращая правильный статус и объект с обновленными данными.

15 Тест: editMedicationSchedule_ShouldReturnNotFound

Описание:

Этот тест проверяет, что при попытке редактировать расписание, которое не существует в системе (например, из-за неправильного или несуществующего идентификатора), система возвращает статус 404 Not Found.

Область тестирования:

— Редактирование расписания, которое не существует.

— Взаимодействие с сервисом MedicationScheduleService при попытке редактировать несуществующее расписание.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для расписания, которое предполагается редактировать.
2. Создается объект MedicationScheduleRequest, который содержит данные для редактирования (например, название медикамента, дни и время приема).
3. Мокируется поведение метода medicationScheduleService.editSchedule(id, request) для того, чтобы он возвращал null, что означает, что расписание с таким идентификатором не найдено в системе.
4. Вызывается метод editSchedule(id, request) контроллера medicationScheduleController, который должен вернуть ошибку, если расписание не найдено.
5. Проверяется, что возвращается статус 404 Not Found, подтверждающий, что редактируемое расписание не найдено.

6. Проверяется, что метод medicationScheduleService.editSchedule(id, request) был вызван с правильными параметрами (id и request).

Примерный вывод из теста:

Если запрашиваемое расписание не существует в системе, система должна вернуть статус 404 Not Found. Метод редактирования расписания должен быть вызван и корректно обработать ситуацию, когда расписание не найдено.

Вывод:

Этот тест удостоверяется, что система корректно обрабатывает ошибку, когда расписание для редактирования не найдено, и возвращает правильный статус ошибки 404 Not Found.

16 Тест: deleteMedicationSchedule_ShouldReturnNoContent

Описание:

Этот тест проверяет, что при удалении расписания медикаментов система возвращает статус 204 No Content, если удаление прошло успешно.

Область тестирования:

— Удаление расписания приема медикаментов.

— Взаимодействие с сервисом MedicationScheduleService для удаления расписания.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для расписания, которое будет удалено.
2. Мокируется поведение метода medicationScheduleService.deleteSchedule(id), чтобы он не вызывал ошибок при удалении и выполнялся успешно (с помощью doNothing()).
3. Вызывается метод deleteSchedule(id) контроллера medicationScheduleController, который должен инициировать удаление расписания по переданному идентификатору.
4. Проверяется, что возвращается статус 204 No Content, что подтверждает успешное удаление без возвращаемого тела.
5. Проверяется, что метод medicationScheduleService.deleteSchedule(id) был вызван с правильным параметром id.

Примерный вывод из теста:

Если расписание было успешно удалено, система должна вернуть статус 204 No Content и не содержать тела ответа. Метод удаления расписания должен быть вызван и обработать запрос корректно.

Вывод:

Этот тест удостоверяется, что при удалении расписания система корректно выполняет операцию и возвращает правильный статус 204 No Content, что подтверждает успешное выполнение операции удаления.

17 Тест: getSettings_ShouldReturnNotificationStatus

Описание:

Этот тест проверяет, что при запросе настроек системы (например, статуса уведомлений) система возвращает статус 200 OK и правильное значение для параметра уведомлений.

Область тестирования:

— Получение настроек системы.

— Взаимодействие с сервисом SettingsService для получения статуса уведомлений.

Шаги выполнения теста:

1. Мокируется поведение метода settingsService.getNotificationStatus(), чтобы он возвращал значение true, что означает, что уведомления включены.
2. Вызывается метод getSettings() контроллера settingsController, который должен вернуть настройки системы (включая статус уведомлений).
3. Проверяется, что возвращается статус 200 OK, подтверждающий успешное выполнение запроса.
4. Проверяется, что тело ответа не null и что оно содержит правильное значение для ключа "notification" в ответе.
5. Проверяется, что метод settingsService.getNotificationStatus() был вызван.

Примерный вывод из теста:

Если запрос на получение настроек прошел успешно, система должна вернуть статус 200 OK и объект, содержащий правильное значение для уведомлений. Метод settingsService.getNotificationStatus() должен быть вызван корректно.

Вывод:

Этот тест удостоверяется, что система правильно возвращает статус 200 OK с соответствующими настройками уведомлений при запросе настроек.

18 Тест: changePassword_ShouldReturnSuccessMessage

Описание:

Этот тест проверяет, что при успешной смене пароля система возвращает статус 200 OK и сообщение "Password changes successfully".

Область тестирования:

— Изменение пароля пользователя.

— Взаимодействие с сервисом SettingsService для изменения пароля.

Шаги выполнения теста:

1. Создается объект PasswordChangeRequest, в который устанавливается новый пароль (newPassword123).
2. Мокируется поведение метода settingsService.changePassword(request.getPassword()), чтобы он выполнялся без ошибок (с помощью doNothing()).
3. Вызывается метод changePassword(request) контроллера settingsController, который должен изменить пароль пользователя.
4. Проверяется, что возвращается статус 200 OK, подтверждающий успешную смену пароля.
5. Проверяется, что в теле ответа содержится сообщение "Password changes successfully".
6. Проверяется, что метод settingsService.changePassword(request.getPassword()) был вызван с правильным параметром (новым паролем).

Примерный вывод из теста:

Если пароль был успешно изменен, система должна вернуть статус 200 OK и сообщение об успешной смене пароля. Метод для смены пароля должен быть вызван с переданным новым паролем.

Вывод:

Этот тест удостоверяется, что при успешной смене пароля система корректно выполняет операцию и возвращает правильный статус и сообщение.

19 Тест: enableNotification_ShouldReturnSuccessMessage

Описание:

Этот тест проверяет, что при включении уведомлений система возвращает статус 200 OK и сообщение "Notification enables", а также что соответствующий метод сервиса был вызван.

Область тестирования:

— Включение уведомлений в системе.

— Взаимодействие с сервисом SettingsService для обновления статуса уведомлений.

Шаги выполнения теста:

1. Мокируется поведение метода settingsService.setNotificationStatus(true), чтобы он выполнялся без ошибок (с помощью doNothing()).
2. Вызывается метод enableNotification() контроллера settingsController, который должен установить статус уведомлений в true (включить уведомления).
3. Проверяется, что возвращается статус 200 OK, подтверждающий успешное выполнение операции.
4. Проверяется, что в теле ответа содержится сообщение "Notification enables".
5. Проверяется, что метод settingsService.setNotificationStatus(true) был вызван для включения уведомлений.

Примерный вывод из теста:

Если уведомления были успешно включены, система должна вернуть статус 200 OK и сообщение "Notification enables". Метод для включения уведомлений должен быть вызван и обработан корректно.

Вывод:

Этот тест удостоверяется, что при включении уведомлений система корректно выполняет операцию и возвращает правильный статус и сообщение.

20 Тест: disableNotification_ShouldReturnSuccessMessage

Описание:

Этот тест проверяет, что при отключении уведомлений система возвращает статус 200 OK и сообщение "Notification disables", а также что соответствующий метод сервиса был вызван для установки статуса уведомлений в false.

Область тестирования:

— Отключение уведомлений в системе.

— Взаимодействие с сервисом SettingsService для обновления статуса уведомлений.

Шаги выполнения теста:

1. Мокируется поведение метода settingsService.setNotificationStatus(false), чтобы он выполнялся без ошибок (с помощью doNothing()).
2. Вызывается метод disableNotification() контроллера settingsController, который должен установить статус уведомлений в false (отключить уведомления).
3. Проверяется, что возвращается статус 200 OK, подтверждающий успешное выполнение операции.
4. Проверяется, что в теле ответа содержится сообщение "Notification disables".
5. Проверяется, что метод settingsService.setNotificationStatus(false) был вызван для отключения уведомлений.

Примерный вывод из теста:

Если уведомления были успешно отключены, система должна вернуть статус 200 OK и сообщение "Notification disables". Метод для отключения уведомлений должен быть вызван и обработан корректно.

Вывод:

Этот тест удостоверяется, что при отключении уведомлений система корректно выполняет операцию и возвращает правильный статус и сообщение.

21 Тест: getMedicalCard_ShouldReturnMedicalCardResponse

Описание:

Этот тест проверяет, что при запросе медицинской карты для пользователя система возвращает корректный объект MedicalCardResponse с правильными данными.

Область тестирования:

— Получение данных медицинской карты пользователя.

— Взаимодействие с сервисом MedicalCardService для получения данных медицинской карты.

— Взаимодействие с репозиторием MedicalCardRepository и сервисом UserService для извлечения данных.

Шаги выполнения теста:

1. Создается объект User, представляющий пользователя, для которого будет запрашиваться медицинская карта.
2. Создается объект MedicalCard, который содержит информацию о медицинской карте пользователя (например, полное имя, рост, вес, группа крови, аллергии, заболевания).
3. Мокируется поведение метода userService.getUserByEmail("test@example.com"), чтобы он возвращал объект user.
4. Мокируется поведение метода medicalCardRepository.findByUser(user), чтобы он возвращал объект medicalCard (внутри Optional).

5. Вызывается метод medicalCardService.getMedicalCard(), который должен извлечь и вернуть данные из медицинской карты.
6. Проверяется, что в ответе возвращаются правильные значения:

— FullName должен быть равен "Test User".

— Height должен быть равен 180.

— Weight должен быть равен 80.

— BloodType должен быть равен "B+".

— Allergies должно быть равно "None".

— Diseases должно быть равно "None"

Примерный вывод из теста:

Если запрос медицинской карты пользователя был успешным, система должна вернуть объект MedicalCardResponse, содержащий все верные данные. Метод получения медицинской карты должен работать правильно, извлекая данные из репозитория и возвращая правильный ответ.

Вывод:

Этот тест удостоверяется, что при запросе медицинской карты для пользователя система правильно возвращает все соответствующие данные в объекте MedicalCardResponse.

22 Тест: editMedicalCard_ShouldUpdateAndReturnMedicalCardResponse

Описание:

Этот тест проверяет, что при редактировании медицинской карты пользователя система обновляет данные в базе и возвращает правильный объект MedicalCardResponse с обновленными значениями.

Область тестирования:

Редактирование медицинской карты пользователя. Взаимодействие с сервисом MedicalCardService для обновления данных медицинской карты. Взаимодействие с репозиторием MedicalCardRepository для сохранения обновленной медицинской карты.

Шаги выполнения теста:

1. Создается объект User, который будет связан с медицинской картой.
2. Создается объект MedicalCard, представляющий текущие данные медицинской карты пользователя.
3. Создается объект MedicalCardRequest, который содержит новые данные для медицинской карты (например, имя, рост, вес, группа крови, аллергии и заболевания).
4. Мокируются следующие действия:

— Метод userService.getUserByEmail("test@example.com) возвращает объект user.

— Метод medicalCardRepository.findByUser(user) возвращает объект Optional.of(medicalCard) с текущими данными медицинской карты.

— Метод medicalCardRepository.save(any(MedicalCard.class)) сохраняет обновленную карту и возвращает ее.

5. Вызывается метод medicalCardService.editMedicalCard(request), который должен обновить данные в медицинской карте и вернуть объект MedicalCardResponse с новыми значениями.
6. Проверяется, что в ответе возвращаются правильные обновленные значения:

— FullName должен быть равен "Test Person".

— Height должен быть равен 180.

— Weight должен быть равен 75.

— BloodType должен быть равен "A+".

— Allergies должно быть равно "Pollen".

— Diseases должно быть равно "Asthma".

7. Проверяется, что метод medicalCardRepository.save(medicalCard) был вызван для сохранения обновленных данных.

Примерный вывод из теста:

Если редактирование медицинской карты прошло успешно, система должна вернуть объект MedicalCardResponse с обновленными данными. Метод для сохранения медицинской карты должен быть вызван, и данные должны быть обновлены в базе.

Вывод:

Этот тест удостоверяется, что при редактировании медицинской карты система корректно обновляет данные и возвращает правильный объект с обновленными значениями.

23 Тест: editMedicalCard_ShouldThrowException_WhenMedicalCardNotFound

Описание:

Этот тест проверяет, что при попытке редактирования медицинской карты, если карта не найдена, система выбрасывает исключение IllegalArgumentException с соответствующим сообщением.

Область тестирования:

— Обработка ситуации, когда медицинская карта пользователя не существует в системе.

— Проверка корректного выбрасывания исключений при отсутствии данных.

Шаги выполнения теста:

Создается объект User, представляющий пользователя.

Создается объект MedicalCardRequest, который содержит данные для редактирования медицинской карты.

Мокируются следующие действия:

Метод userService.getUserByEmail("test@example.com") возвращает объект user.

Метод medicalCardRepository.findByUser(user) возвращает Optional.empty(), что означает, что медицинская карта для данного пользователя не найдена.

Вызывается метод medicalCardService.editMedicalCard(request), который должен попытаться обновить медицинскую карту.

Проверяется, что при вызове этого метода выбрасывается исключение IllegalArgumentException.

Проверяется, что сообщение исключения соответствует ожидаемому: "Medical card not found".

Примерный вывод из теста:

Если медицинская карта не найдена в системе, метод должен выбросить исключение с корректным сообщением. Это предотвращает попытки редактировать несуществующие записи.

Вывод:

Этот тест удостоверяется, что при попытке редактировать несуществующую медицинскую карту система выбрасывает исключение с правильным сообщением о том, что карта не найдена.

24 Тест: getUserSchedules_ShouldReturnSchedules

Описание:

Этот тест проверяет, что при запросе расписания пользователя система возвращает список медикаментозных расписаний, если такие существуют.

Область тестирования:

— Получение расписаний медикаментов пользователя.

— Взаимодействие с репозиториями UserRepository и MedicationScheduleRepository для получения расписаний.

Шаги выполнения теста:

1. Создается объект MedicationSchedule, который представляет одно расписание с данными медикамента ("Aspirin"), днями расписания (1 и 2) и временем приема (8:00).
2. Мокируются следующие действия:

— Метод userRepository.findByEmail("test@example.com") возвращает объект user.

— Метод medicationScheduleRepository.findByUser(user) возвращает список расписаний (в данном случае, один объект MedicationSchedule).

3. Вызывается метод medicationScheduleService.getUserSchedules(), который должен вернуть список расписаний пользователя.

4. Проверяется, что возвращается список с одним элементом.
5. Проверяется, что первое расписание в списке имеет правильное название медикамента "Aspirin".
6. Проверяется, что метод medicationScheduleRepository.findByUser(user) был вызван.

Примерный вывод из теста:

Если расписание пользователя существует, система должна вернуть правильный список расписаний, содержащий ожидаемое расписание. Метод репозитория для получения расписаний должен быть вызван и обработан корректно.

Вывод:

Этот тест удостоверяется, что при запросе расписаний пользователя система корректно возвращает список с нужными данными, и что методы репозитория были вызваны должным образом.

25 Тест: addSchedule_ShouldAddScheduleSuccessfully

Описание:

Этот тест проверяет, что при добавлении нового расписания для медикамента система успешно сохраняет расписание и возвращает его с правильными данными.

Область тестирования:

— Добавление нового расписания для медикаментов.

— Взаимодействие с репозиторием MedicationScheduleRepository для сохранения расписания.

Шаги выполнения теста:

1. Мокируется поведение метода userRepository.findByEmail("test@example.com"), чтобы он возвращал объект user.
2. Создается объект MedicationSchedule, который содержит информацию о расписании для медикамента "Aspirin", включая дни приема (1, 2, 3) и время приема (8:00 и 18:00).
3. Устанавливается связь с пользователем через метод schedule.setUser(user).
4. Мокируется поведение метода medicationScheduleRepository.save(any(MedicationSchedule.class)), чтобы он возвращал объект schedule при попытке сохранить его.
5. Вызывается метод medicationScheduleService.addSchedule(request), который должен добавить новое расписание и вернуть его.

6. Проверяется, что возвращенное расписание имеет правильное название медикамента "Aspirin".
7. Проверяется, что возвращенное расписание содержит два времени приема (8:00 и 18:00).
8. Проверяется, что метод medicationScheduleRepository.save(any(MedicationSchedule.class)) был вызван для сохранения расписания.

Примерный вывод из теста:

Если новое расписание было успешно добавлено, система должна вернуть объект расписания с правильными данными. Метод сохранения расписания должен быть вызван и обработан корректно.

Вывод:

Этот тест удостоверяется, что при добавлении нового расписания система корректно выполняет операцию сохранения и возвращает правильный объект с ожидаемыми данными.

26 Тест: editSchedule_ShouldUpdateAndReturnSchedule

Описание:

Этот тест проверяет, что при редактировании расписания медикамента система обновляет данные и возвращает обновленное расписание.

Область тестирования:

— Обновление существующего расписания медикамента.

— Взаимодействие с репозиторием MedicationScheduleRepository для сохранения изменений в расписании.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для уже существующего расписания.
2. Создается объект MedicationSchedule с исходными данными, например, с названием медикамента "Aspirin" и привязанным пользователем.
3. Создается объект MedicationScheduleRequest, который содержит новые данные для обновления расписания, такие как новое название медикамента ("Paracetamol"), новые дни расписания (2 и 4) и новое время приема (10:00 и 16:00).
4. Мокируются следующие действия:

— Метод userRepository.findByEmail("test@example.com") возвращает объект user.

— Метод medicationScheduleRepository.findById(scheduleId) возвращает объект existingSchedule с текущими данными расписания.

— Ме-
тод medicationScheduleRepository.save(any(MedicationSchedule.clas
s)) сохраняет обновленное расписание и возвращает его.

5. Вызывается ме-
тод medicationScheduleService.editSchedule(scheduleId, newRequest
), который должен обновить расписание и вернуть обновлен-
ные данные.
6. Проверяется, что название медикамента в обновленном расписа-
нии равно "Paracetamol".
7. Проверяется, что количество дней в расписании равно 2.
8. Проверяется, что первое время из списка расписа-
ния равно 10:00.
9. Проверяется, что ме-
тод medicationScheduleRepository.save(existingSchedule) был вы-
зван для сохранения обновленного расписания.

Примерный вывод из теста:

Если расписание было успешно обновлено, система должна вернуть объект расписания с обновленными данными. Метод сохранения расписания должен быть вызван, и данные должны быть обновлены корректно.

Вывод:

Этот тест удостоверяется, что при редактировании расписания система корректно обновляет данные и возвращает правильный объект с обновленными значениями.

27 Тест: editSchedule_ShouldThrowException_WhenScheduleNotFound

Описание:

Этот тест проверяет, что если попытаться отредактировать расписание, которое не существует (не найдено в базе данных), система выбрасывает исключение IllegalArgumentException с соответствующим сообщением "Schedule not found".

Область тестирования:

— Обработка ситуации, когда пытаемся редактировать расписание, которого нет в базе данных.

— Проверка корректного выбрасывания исключения в случае отсутствия данных.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для несуществующего расписания.
2. Мокируются следующие действия:

— Метод userRepository.findByEmail("test@example.com") возвращает объект user для успешного выполнения поиска пользователя.

— Метод medicationScheduleRepository.findById(scheduleId) возвращает Optional.empty(), что означает, что расписание с указанным идентификатором не найдено.

3. Вызывается метод medicationScheduleService.editSchedule(scheduleId, request), который должен попытаться обновить расписание.

4. Проверяется, что при вызове этого метода выбрасывается исключение IllegalArgumentException.
5. Проверяется, что сообщение исключения соответствует ожидаемому: "Schedule not found".

Примерный вывод из теста:

Если расписание не найдено в базе данных, метод должен выбросить исключение с корректным сообщением. Это предотвращает попытки редактирования несуществующих расписаний и гарантирует, что система правильно обрабатывает такие ситуации.

Вывод:

Этот тест удостоверяется, что при попытке редактировать несуществующее расписание система выбрасывает исключение с правильным сообщением "Schedule not found".

28 Тест: deleteSchedule_ShouldDeleteSuccessfully

Описание:

Этот тест проверяет, что при удалении расписания система корректно удаляет его из базы данных и вызывает метод репозитория для выполнения операции удаления.

Область тестирования:

— Удаление расписания медикаментов.

— Взаимодействие с репозиторием MedicationScheduleRepository для удаления записи.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для расписания, которое будет удалено.
2. Создается объект MedicationSchedule, представляющий расписание, и устанавливаются его данные, включая идентификатор scheduleId и пользователя user.
3. Мокируются следующие действия:

— Метод userRepository.findByEmail("test@example.com") возвращает объект user, подтверждая, что пользователь существует.

— Метод medicationScheduleRepository.findByIdAndUser(scheduleId, user) возвращает объект schedule, который представляет существующее расписание.

4. Вызывается ме-
   тод medicationScheduleService.deleteSchedule(scheduleId), кото-
   рый должен удалить расписание.
5. Проверяется, что ме-
   тод medicationScheduleRepository.delete(schedule) был вы-
   зван для удаления расписания.

Примерный вывод из теста:

Если расписание было успешно удалено, система должна вызвать метод удаления на репозитории, и операция должна быть завершена без ошибок.

Вывод:

Этот тест удостоверяется, что при удалении расписания система правильно взаимодействует с репозиторием и удаляет расписание из базы данных.

29 Тест: deleteSchedule_ShouldThrowException_WhenScheduleNotFound

Описание:

Этот тест проверяет, что если при попытке удалить расписание оно не найдено или не принадлежит пользователю, система выбрасывает исключение IllegalArgumentException с соответствующим сообщением "Schedule not found or not owned by user".

Область тестирования:

— Обработка ситуации, когда расписание не найдено или не принадлежит пользователю.

— Проверка корректного выбрасывания исключений при отсутствии данных.

Шаги выполнения теста:

1. Создается уникальный идентификатор UUID для несуществующего расписания.
2. Мокируются следующие действия:

— Метод userRepository.findByEmail("test@example.com") возвращает объект user.

— Метод medicationScheduleRepository.findByIdAndUser(scheduleId, user) возвращает null, что означает, что расписание с указанным идентификатором либо не найдено, либо не принадлежит пользователю.

3. Вызывается ме-
   тод medicationScheduleService.deleteSchedule(scheduleId), кото-
   рый должен попытаться удалить расписание.
4. Проверяется, что при вызове этого метода выбрасывается исклю-
   чение IllegalArgumentException.
5. Проверяется, что сообщение исключения соответствует ожидае-
   мому: "Schedule not found or not owned by user".

Примерный вывод из теста:

Если расписание не найдено или не принадлежит пользователю, ме-
тод должен выбросить исключение с корректным сообщением. Это предот-
вращает попытки удаления несуществующих расписаний или расписаний, ко-
торые не принадлежат пользователю.

Вывод:

Этот тест удостоверяется, что при попытке удалить несуществую-
щее расписание или расписание, которое не принадлежит пользователю, си-
стема выбрасывает исключение с правильным сообще-
нием "Schedule not found or not owned by user".

30 Тест: generateCode_ShouldReturnCode

Описание:

Этот тест проверяет, что при вызове метода генерации кода для восстановления пароля система возвращает код длиной 6 символов, состоящий исключительно из цифр.

Область тестирования:

— Генерация кода для восстановления пароля.

— Проверка, что возвращаемый код соответствует заданным требованиям (длина 6 символов и только цифры).

Шаги выполнения теста:

1. Вызывается метод passwordRecoveryService.generateCode(), который должен сгенерировать шестизначный код для восстановления пароля.
2. Проверяется, что длина возвращаемого кода равна 6.
3. Проверяется, что код состоит исключительно из цифр, используя регулярное выражение \\d{6} (что означает 6 цифр).

Примерный вывод из теста:

— Если код имеет правильную длину и состоит из цифр, тест должен пройти успешно.

— Это гарантирует, что метод генерации кода возвращает корректные данные, соответствующие требованиям.

Вывод:

Этот тест удостоверяется, что метод генерации кода для восстановления пароля всегда возвращает правильный код, состоящий из 6 цифр.

31 Тест: sendRecoveryCode_ShouldSendEmailAndStoreCode

Описание:

Этот тест проверяет, что при запросе на восстановление пароля, система генерирует код для восстановления, отправляет его на email и сохраняет код для последующей проверки.

Область тестирования:

— Генерация кода для восстановления пароля.

— Отправка email с кодом восстановления.

— Сохранение кода в системе для дальнейшей верификации.

Шаги выполнения теста:

1. Мокируется поведение метода userRepository.findByEmail("test@example.com"), чтобы он возвращал объект user, который соответствует email, с которого будет запрашиваться восстановление пароля.
2. Мокируется метод mailSender.createMimeMessage() для создания MimeMessage, который представляет собой email-сообщение.
3. Используется ReflectionTestUtils для установки значения поля username в объекте passwordRecoveryService на "noreply@example.com" — это адрес отправителя email.
4. Генерируется код для восстановления пароля с помощью passwordRecoveryService.generateCode().

5. Вызывается ме-
   тод passwordRecoveryService.sendRecoveryCode("test@example.co
   m", code), который должен отправить сгенерирован-
   ный код на указанный email.
6. Проверяется, что метод mailSender.send() был вы-
   зван ровно один раз для отправки email.
7. Проверяется, что код был сохранен в системе и его можно прове-
   рить с помощью ме-
   тода passwordRecoveryService.verifyCode("test@example.com", cod
   e).

Примерный вывод из теста:

Если код был успешно сгенерирован, отправлен на email и сохранен, то тест пройдет успешно. Это гарантирует, что процесс восстановления пароля работает корректно, включая отправку email и сохранение кода для дальнейшей верификации.

Вывод:

Этот тест удостоверяется, что при отправке кода восстановления пароля система корректно генерирует код, отправляет его по email и сохраняет его для дальнейшей проверки.

32 Тест: sendRecoveryCode_ShouldThrowException_WhenUserNotFound

Описание:

Этот тест проверяет, что если при попытке отправить код для восстановления пароля система не может найти пользователя по указанному email, то выбрасывается исключение RuntimeException.

Область тестирования:

— Обработка ошибок при отправке кода восстановления пароля.

— Проверка корректного выбрасывания исключений, если пользователь не найден.

Шаги выполнения теста:

1. Мокируется метод userRepository.findByEmail("notfound@example.com"), чтобы он возвращал Optional.empty(), что означает, что пользователь с таким email не найден в системе.
2. Генерируется код для восстановления пароля с помощью метода passwordRecoveryService.generateCode().
3. Вызывается метод passwordRecoveryService.sendRecoveryCode("notfound@example.com", code), который должен попытаться отправить код на email.
4. Проверяется, что при вызове этого метода выбрасывается исключение RuntimeException, поскольку пользователь с указанным email не найден в базе данных.

Примерный вывод из теста:

Если пользователь не найден, метод должен выбросить исключение, чтобы предотвратить попытку отправки кода на несуществующий email. Этот тест помогает гарантировать, что система правильно обрабатывает ошибку при отсутствии пользователя в базе данных.

Вывод:

Этот тест удостоверяется, что при попытке отправить код для восстановления пароля на несуществующий email система выбрасывает исключение RuntimeException.

33 Тест: verifyCode_ShouldReturnFalse_WhenCodeExpired

Описание:

Этот тест проверяет, что при попытке верификации кода для восстановления пароля после истечения его срока действия система возвращает false. Это гарантирует, что устаревшие коды не могут быть использованы для восстановления пароля.

Область тестирования:

— Проверка срока действия кода для восстановления пароля.

— Обработка истечения срока действия кода и предотвращение его использования после истечения времени.

Шаги выполнения теста:

1. Мокируются следующие действия:

— Метод userRepository.findByEmail("test@example.com") возвращает объект user.

— Метод mailSender.createMimeMessage() создает объект MimeMessage для имитации отправки письма.

2. Генерируется код для восстановления пароля с помощью метода passwordRecoveryService.generateCode().

3. Вызывается метод passwordRecoveryService.verifyCode("test@example.com", code), который проверяет правильность кода.

4. Извлекается внутреннее поле codeTimestamps с помощью ReflectionTestUtils для получения временных меток кодов восстановления.

5. Время для указанного email обновляется в timestamps.put("test@example.com", ...), чтобы имитировать истечение срока действия кода (устанавливается время в прошлом, например, 16 минут назад).
6. Снова вызывается метод passwordRecoveryService.verifyCode("test@example.com", code) для верификации кода.
7. Проверяется, что результат верификации равен false, что означает, что код истек и не может быть использован для восстановления пароля.

Примерный вывод из теста:

Если код истек, система должна вернуть false при попытке его верификации. Это помогает гарантировать, что старые или просроченные коды не могут быть использованы для восстановления пароля.

Вывод:

Этот тест удостоверяется, что система корректно обрабатывает истекшие коды для восстановления пароля и предотвращает их использование.

34 Тест: verifyCode_ShouldReturnFalse_WhenNoCodeStored

Описание:

Этот тест проверяет, что если код для восстановления пароля не был ранее сохранен в системе для данного пользователя, метод верификации должен вернуть false.

Область тестирования:

— Проверка, что система возвращает false, если код для восстановления пароля не был найден для пользователя.

— Обработка ситуации, когда код не существует в хранилище.

Шаги выполнения теста:

1. Вызывается метод passwordRecoveryService.verifyCode("test@example.com", "123456"), который пытается проверить код для восстановления пароля для пользователя с email "test@example.com".
2. Поскольку код для этого email не был сохранен в системе (предполагается, что код еще не был сгенерирован или сохранен), метод должен вернуть false.
3. Проверяется, что результат вызова метода verifyCode равен false.

Примерный вывод из теста:

Если код не был сохранен для пользователя, метод верификации должен возвращать false, предотвращая неправильное использование кода.

Вывод:

Этот тест удостоверяется, что система корректно возвращает false при попытке верификации кода, если код для восстановления пароля не был сохранен для указанного пользователя.

35 Тест: resetPassword_ShouldChangePasswordAndClearCodes

Описание:

Этот тест проверяет, что при успешном сбросе пароля система корректно изменяет пароль пользователя и очищает сохраненные коды восстановления, чтобы они не могли быть использованы повторно.

Область тестирования:

— Сброс пароля пользователя.

— Обновление пароля с использованием метода кодирования пароля.

— Очищение сохраненных кодов для восстановления пароля и временных меток после успешного сброса.

Шаги выполнения теста:

1. Мокируется метод userRepository.findByEmail("test@example.com"), чтобы он возвращал объект user.
2. Мокируется метод passwordEncoder.encode("newPassword"), чтобы он возвращал строку "encodedPassword" (симулируя успешное кодирование нового пароля).
3. Генерируется код восстановления пароля с помощью метода passwordRecoveryService.generateCode().
4. Извлекаются внутренние поля recoveryCodes и codeTimestamps с помощью ReflectionTestUtils для доступа к сохраненным кодам и временным меткам.

5. Добавляется сгенерированный код и текущая временная метка для email "test@example.com" в recoveryCodes и codeTimestamps.
6. Вызывается метод passwordRecoveryService.resetPassword("test@example.com", "newPassword"), который должен:

— Изменить пароль пользователя на "newPassword" (закодированный как "encodedPassword").

— Очистить код восстановления и временную метку для указанного email.

7. Проверяется, что пароль пользователя был изменен на "encodedPassword".
8. Проверяется, что метод userRepository.save(user) был вызван для сохранения изменений.
9. Проверяется, что код восстановления и временная метка были удалены из recoveryCodes и codeTimestamps для указанного email.

Примерный вывод из теста:

Если сброс пароля был успешным, пароль пользователя должен быть обновлен, и все коды восстановления и временные метки должны быть очищены. Это гарантирует, что после сброса пароля старые коды не могут быть использованы повторно.

Вывод:

Этот тест удостоверяется, что при сбросе пароля система корректно обновляет пароль и очищает все связанные коды восстановления и временные метки, обеспечивая безопасность данных.

36 Тест: changePassword_ShouldEncodeAndSaveNewPassword

Описание:

Этот тест проверяет, что при смене пароля система корректно кодирует новый пароль и сохраняет его в базе данных.

Область тестирования:

— Смена пароля пользователя.

— Кодирование пароля с помощью passwordEncoder.

— Сохранение обновленного пароля в базе данных.

Шаги выполнения теста:

1. Создается строка newPassword с новым паролем ("newPassword"), который пользователь хочет установить.
2. Создается строка encodePassword, которая представляет закодированное значение пароля (например, "encodePassword").
3. Мокируется поведение метода userService.getUserByEmail("test@example.com"), чтобы он возвращал объект user, который является текущим пользователем.
4. Мокируется метод passwordEncoder.encode(newPassword) для возврата закодированного пароля ("encodePassword").
5. Проверяется, что пароль пользователя был изменен на закодированное значение (encodePassword).
6. Проверяется, что метод userRepository.save(user) был вызван для сохранения изменений в базе данных.

Примерный вывод из теста:

Если смена пароля прошла успешно, пароль пользователя должен быть изменен на закодированное значение. Метод сохранения пользователя должен быть вызван для обновления данных в базе.

Вывод:

Этот тест удостоверяется, что при смене пароля система правильно кодирует новый пароль и сохраняет его в базе данных.

37 Тест: getNotificationStatus_ShouldReturnCurrentStatus

Описание:

Этот тест проверяет, что метод получения статуса уведомлений корректно возвращает текущий статус для пользователя.

Область тестирования:

— Получение текущего статуса уведомлений для пользователя.

— Взаимодействие с сервисом UserService для получения информации о пользователе.

Шаги выполнения теста:

1. Мокируется метод userService.getUserByEmail("test@example.com"), чтобы он возвращал объект user с данными пользователя, для которого будет запрашиваться статус уведомлений.
2. Вызывается метод settingsService.getNotificationStatus(), который должен вернуть текущий статус уведомлений для указанного пользователя.
3. Проверяется, что результат выполнения метода getNotificationStatus() равен true, что означает, что уведомления включены для этого пользователя.
4. Проверяется, что метод userService.getUserByEmail("test@example.com") был вызван для получения данных пользователя.

Примерный вывод из теста:

Если уведомления включены для пользователя, метод должен вернуть true, что означает, что уведомления активированы для пользователя.

Вывод:

Этот тест удостоверяется, что метод получения статуса уведомлений корректно возвращает текущий статус уведомлений для указанного пользователя.

38 Тест: setNotificationStatus_ShouldUpdateAndSaveStatus

Описание:

Этот тест проверяет, что при изменении статуса уведомлений система корректно обновляет этот статус и сохраняет изменения в базе данных.

Область тестирования:

— Обновление статуса уведомлений.

— Сохранение изменений в базе данных с помощью метода userRepository.save().

Шаги выполнения теста:

1. Мокируется метод userService.getUserByEmail("test@example.com"), чтобы он возвращал объект user для пользователя, чей статус уведомлений будет обновляться.
2. Вызывается метод settingsService.setNotificationStatus(false), который должен установить статус уведомлений в значение false.
3. Проверяется, что статус уведомлений был обновлен на false с помощью метода settingsService.getNotificationStatus().
4. Проверяется, что метод userRepository.save(user) был вызван, подтверждая, что изменения были сохранены в базе данных.

Примерный вывод из теста:

Если статус уведомлений был успешно обновлен, метод должен вернуть false, подтверждая, что уведомления были отключены. Метод сохранения пользователя должен быть вызван для обновления данных в базе.

Вывод:

Этот тест удостоверяется, что при изменении статуса уведомлений система правильно обновляет этот статус и сохраняет изменения в базе данных.

Тестирование Frontend

1. **Тест: Тестирование адаптивности (Responsive Design)**

Цель:

Проверить, приложение адаптируется под разные размеры экранов мобильных устройств.

Шаги:

— Открыли страницу с формой регистрации или авторизации.

— Открыли приложение на мобильном устройстве.

— Уменьшили и увеличили размер экрана, чтобы проверить корректную адаптацию интерфейса.

— Убедились, что элементы интерфейса (текст, кнопки, изображения) корректно масштабируются, не выходят за пределы экрана, и сохраняют правильную читаемость.

— Проверили, что меню и другие элементы управления адаптируются в зависимости от размера экрана.

Результат:

— Все элементы интерфейса адекватно адаптируются и правильно расположены на мобильных экранах.

— Меню и другие элементы управления удобные для использования на мобильных устройствах.

2. Тест: Тестирование касания (Touch Interactions)

Цель:

Проверить, что все элементы, которые взаимодействуют с пользователем, корректно обрабатывают касания.

Шаги:

— Протестировали все кнопки, поля ввода и переключатели на мобильном устройстве.

— Убедились, что кнопки нажимаются при касании, а при свайпе происходит правильная реакция.

— Убедились, что все элементы, требующие ввода данных, работают корректно на сенсорном экране.

Результат:

— Все элементы реагируют на касание или свайп, работают плавно и без задержек.
— Пользователь легко взаимодействует с приложением без проблем с распознаванием касания.

3. **Тест: Тестирование производительности (Performance Testing)**

Цель:

Проверить, как приложение работает на мобильных устройствах с ограниченными ресурсами.

Шаги:

— Открыли приложение на мобильном устройстве и проверили скорость загрузки.

— Протестировали анимации, эффекты, плавность прокрутки.

— Проверили, что приложение не тормозит или зависает при работе с большим количеством данных.

Результат:

— Приложение запускается быстро, а взаимодействие с приложением плавное и без задержек.

4. **Тест: Тестирование работы формы (Form Testing)**

Цель:

Проверили работу всех форм на мобильных устройствах.

Шаги:

— Открыли форму регистрации, входа и другие формы на мобильном устройстве.

— Прове-
рили, что все поля формы видны, удобны для ввода, и имеют правильную разметку.

— Проверили клавиатуру на мобильном устройстве (например, для ввода email, номера телефона, пароля).

Результат:

— Все поля видны и корректно работают при вводе данных.

Нагрузочный тест

Цель теста:

Проверка устойчивости и производительности системы при одновременной нагрузке с использованием виртуальных пользователей для имитации реальных пользователей.

Тестовые параметры:

— Пользователи: 20 виртуальных пользователей.

— Продолжительность: 5 минут 30 секунд.

— Всего запросов: 2 567 запросов.

— Средняя скорость обработки запросов: 8.3 запроса в секунду.

— Максимальная нагрузка: 14 запросов в секунду.

— Среднее время ответа: 887 мс 95% запросов уложились в: 1.3 секунды.

— Максимальное время ответа: 4.79 секунды.

— Ошибок: 0 (все запросы были успешными).

— Пропускная способность: 1.23 МБ получено, 294 КБ отправлено.

Результаты:

Тест показал хорошие результаты. Все запросы были обработаны без ошибок, время отклика находилось в пределах нормы, а система смогла выдержать одновременную нагрузку в 20 виртуальных пользователей. Среднее время ответа и максимальное время отклика также находятся в приемлемых пределах, что свидетельствует о хорошей производительности системы. Пропускная способность сети также была достаточно высокой.

Вывод:

Тест прошёл успешно, и система стабильно выдерживает текущую нагрузку. Однако стоит отметить, что 20 виртуальных пользователей — это не конечный результат, а ограничение тестирования. Планируется провести дальнейшее тестирование с увеличением количества пользователей, чтобы убедиться, что система будет работать стабильно при увеличении реальной нагрузки.