# Тестовое задание. Дудкин Александр

## Задание №1 Проверĸа мини-приложения бронирования столиĸа

Дано:  
Форма бронирования состоит из следующих полей:  
• Имя – текстовое поле, обязательное.  
• Количество гостей – числовое поле, от 1 до 10.  
• Дата брони – выбирается из календаря. Можно забронировать только на будущие даты, максимум за 30 дней вперёд.  
• Время – выбирается из выпадающего списка (с 12:00 до 22:00, с шагом 30 мин).  
• Комментарий – необязательное поле (до 250 символов).  
• Кнопка “Забронировать” – активна, когда обязательные поля заполнены корректно.

1.1. Анализ и уточнение:

Возможные противоречия:

* Неясно, как система будет обрабатывать бронирования, когда могут быть ограничения или исключения в доступности столиков.
* Какие ограничения действуют на время бронирования (например, нельзя ли забронировать столик на текущую дату или уже прошедшее время)?
* Не указано, какие сообщения об ошибках будут отображаться в случае некорректного ввода данных.
* Не описана возможность отмены бронирования

Уточняющие вопросы к аналитику/менеджеру:

* Из всех полей только **Имя** отмечено, как обязательное и **Комментарий**, как необязательное. Просьба уточнить какие поля являются обязательными для заполнения.
* Должен ли указываться конкретный столик для бронирования? Так как пользователю может быть важно расположение столика.
* **Количество гостей** – Как будет обрабатываться ситуация, если пользователь введет неверное количество гостей (например, 0 или больше 10)?
* **Дата брони** - Как должны помечаться пользователю доступные для бронирования даты? Не ясно, как система будет обрабатывать ситуации, когда пользователь пытается забронировать столик на более чем 30 дней вперед.
* **Время** - Сколько времени затрачивается на бронирование строго определено или должен указываться временной промежуток?
* Какие сообщения об ошибках и валидации будут отображаться, если обязательные поля не заполнены или заполнены некорректно?
* Какой функционал должен быть доступен для пользователей, которые хотят отменить или изменить бронирование?

1.2. Тест-кейсы

1. Тест-кейс 1: Корректное бронирование

- Описание: Проверка успешного бронирования с корректными данными.

- Шаги:

1. Ввести имя "Иван".

2. Ввести число гостей 4.

3. Выбрать дату 15 дней вперёд.

4. Выбрать время 14:00.

5. Нажать "Забронировать".

- Ожидаемый результат: Сообщение об успешном бронировании.

2. Тест-кейс 2: Бронирование с комментарием

- Описание: Проверка успешного бронирования с добавлением комментария.

- Шаги:

1. Ввести имя "Елена".

2. Ввести число гостей 2.

3. Выбрать дату 10 дней вперёд.

4. Выбрать время 18:00.

5. Ввести комментарий "День рождения".

6. Нажать "Забронировать".

- Ожидаемый результат: Сообщение об успешном бронировании.

3. Тест-кейс 3: Неверное количество гостей

- Описание: Проверка, что система не позволяет забронировать столик на 0 гостей.

- Шаги:

1. Ввести имя "Алекс".

2. Ввести число гостей 0.

3. Выбрать дату 20 дней вперёд.

4. Выбрать время 19:00.

5. Нажать "Забронировать".

- Ожидаемый результат: Сообщение об ошибке: "Количество гостей должно быть от 1 до 10".

4. Тест-кейс 4: Бронирование на прошедшую дату

- Описание: Проверка, что невозможно выбрать прошедшую дату.

- Шаги:

1. Ввести имя "Ольга".

2. Ввести число гостей 3.

3. Выбрать дату, которая была вчера.

4. Выбрать время 17:00.

5. Нажать "Забронировать".

- Ожидаемый результат: Сообщение об ошибке: "Выберите будущую дату".

5. Тест-кейс 5: Превышение максимального числа символов в комментарии

- Описание: Проверка, что невозможно ввести больше 250 символов в поле комментария.

- Шаги:

1. Ввести имя "Максим".

2. Ввести число гостей 2.

3. Выбрать дату 5 дней вперёд.

4. Выбрать время 12:30.

5. Ввести комментарий длиной 300 символов.

6. Нажать "Забронировать".

- Ожидаемый результат: Сообщение об ошибке: "Комментарий не может превышать 250 символов".

1.3. Баги

1. Баг 1: Можно выбрать текущую дату и уже прошедшее время

**Описание:** Пользователь может выбрать текущую дату и прошедшее время.

**Приоритет:** P1 (Критический)

**Обоснование:** Позволяет пользователям делать некорректные бронирования, что нарушает бизнес-логику приложения.

**Шаги:**

1. Заполнить обязательные поля(Имя)
2. В поле **Дата брони** ввести текущую дату
3. В поле **Время** выбрать время меньше текущего
4. Нажать кнопку **Забронировать**

**ФР:** Система дает забронировать выбрав текущую дату и уже прошедшее время.

**ОР:** Появляется сообщение о некорректном вводе даты и времени.

2. Баг 2: В поле “Имя” можно ввести только 10 символов

**Описание:** Ограничение на количество символов в поле имени.

**Приоритет:** P2 (Высокий)

**Обоснование:** Чтобы обеспечить полную идентификацию пользователя, стоит позволить вводить более длинные имена, если таковы нужны.

**Шаги:**

1. Ввести в поле **Имя** значение больше 10 символов. Например, Пантелеймон
2. Нажать кнопку **Забронировать**

**ФР:** Появляется ошибка бронирования

**ОР:** В поле “Имя” можно ввести больше 10 символов

3. Баг 3: Кнопка “Забронировать” активна даже при пустом поле "Имя"

**Описание:** Кнопка доступна для нажатия, даже если поле "Имя" пустое.

**Приоритет:** P1 (Критический)

**Обоснование:** Это приводит к возможным ошибкам при бронировании и нарушает логику валидации формы.

**Шаги:**

1. Поле **Имя** оставить незаполненным
2. Обратить внимание на кнопку **Забронировать**

**ФР:** Кнопка **Забронировать** активна даже при пустом поле **Имя**

**ОР:** Кнопка **Забронировать** не активна при пустом поле **Имя**

4. Баг 4: Поле “Комментарий” не имеет ограничения — можно ввести 1000+ символов

**Описание:** Пользователь может ввести больше 250 символов в поле комментария.

**Приоритет:** P2 (Высокий)

**Обоснование:** Это может привести к проблемам с отображением данных и сложности обработки на сервере.

**Шаги:**

1. Заполнить обязательные поля
2. В поле **Комментарий** ввести больше 1000 символов

**ФР:** В поле **Комментарий** можно ввести 1000+ символов

**ОР:** В поле **Комментарий** можно ввести меньше 250 символов

1.4. Оценка UX и улучшения

1. Удобство пользователя (UX):

Улучшение интерфейса: Сделать форму более интерактивной, добавив подсказки и уведомления, которые будут предупреждать пользователя о необходимых полях перед отправкой формы.

2. Предотвращение ошибок:

Валидация данных в реальном времени: Реализовать проверку обязательных полей и диапазона значений (например, число гостей) до нажатия кнопки "Забронировать", чтобы предотвратить отправку некорректной информации.

3. Повышение качества продукта:

Внедрение системы автоматического тестирования: Разработать набор автоматизированных тестов для проверки функциональности формы и валидации ввода, чтобы гарантировать корректность работы всех функций приложения.

## Задание №2 SQL

2.1. Напиши SQL-запрос для выборки всех активных пользователей старше 25 лет.

SELECT \*

FROM USERS

WHERE status = 'active' AND age > 25;

2.2. Как получить количество неактивных пользователей?

SELECT COUNT(\*) AS inactive\_user\_count

FROM USERS

WHERE status = 'inactive';

2.3. Какой запрос покажет имена и email всех пользователей, отсортированных по возрасту по убыванию?

SELECT name, email

FROM USERS

ORDER BY age DESC;

2.4. Как бы ты проверил, что при удалении пользователя данные из связанных таблиц (например, orders) тоже удаляются корректно?

- Использовать каскадное удаление (ON DELETE CASCADE), если есть внешние ключи, определенные на уровне структуры базы данных. Это гарантирует, что при удалении записи из таблицы test все связанные записи в таблице orders (или других связанных таблицах) также будут удалены.

Пример создания внешнего ключа с каскадным удалением:

CREATE TABLE orders (

order\_id INTEGER PRIMARY KEY,

user\_id INTEGER,

FOREIGN KEY (user\_id) REFERENCES USERS(id) ON DELETE CASCADE

);

- После этого как добавятся данные, можно удалить пользователя:

DELETE FROM USERS WHERE id = “нужный ID пользователя”

- Затем выполнить запрос на выборку из таблицы orders по пользователю, чтобы убедиться, что связанные данные были удалены.

SELECT \* FROM orders WHERE user\_id = “нужный ID пользователя”

2.5. Придумай 2 бага, которые можно было бы найти, проверяя данные через SQL

1. Некорректное значение возраста:

- Если в таблице есть пользователи с возрастом ниже 0 или с нечисловыми значениями, это является ошибкой. Можно выполнить запрос для нахождения таких пользователей:

SELECT \* FROM USERS WHERE age < 0 OR age IS NULL;

2. Если в системе email пользователей должны быть уникальными, то можно проверить наличие дубликатов:

SELECT email, COUNT(\*)

FROM USERS

GROUP BY email

HAVING COUNT(\*) > 1;

## Задание №3 API

| Контекст:  Ты тестируешь API для создания пользователя:  **POST /api/v1/users**  Пример тела запроса:  {  "name": "Анна",  "email": "anna@example.com",  "age": 27  }  Ожидается, что при корректном вводе API возвращает:  {  "id": 123,  "name": "Анна",  "email": "anna@example.com",  "age": 27,  "status": "active"  } | 1 Какие поля ты бы проверил на валидацию? Приведи 3 примера некорректных запросов.  2 Что бы ты проверил в заголовках (headers) запроса и ответа?  3 Какие типы тестирования можно проводить через Postman?  4 Какие коды ответа (status codes) ты ожидаешь при:  ◦ Успешном создании  ◦ Ошибке валидации  ◦ Попытке создать пользователя с уже существующим email  5 Как бы ты оформил тест-кейс на проверку этого эндпоинта? |
| --- | --- |

3.1. Поля для проверки на валидацию и примеры некорректных запросов

Поля для валидации:

- name: Должен быть строкой и не должен быть пустым.

- email: Должен соответствовать формату email и быть уникальным.

- age: Должен быть целым числом и находиться в допустимом диапазоне (например, от 1 до 120).

Примеры некорректных запросов:

3.1.1. Пустое имя:

{

"name": "",

"email": "anna@example.com",

"age": 27

}

3.1.2. Некорректный формат email:

{

"name": "Анна",

"email": "annaexample.com", // неправильный формат

"age": 27

}

3.1.3. Возраст вне допустимого диапазона:

{

"name": "Анна",

"email": "anna@example.com",

"age": -5 // некорректное значение

}

3.2. Проверка заголовков (headers)

Заголовки запроса:

- Content-Type: Убедиться, что установленный тип содержимого (обычно application/json для JSON-запросов).

- Authorization: Если API требует аутентификации, убедитесь, что заголовок аутентификации присутствует и корректный.

Заголовки ответа:

- Content-Type: Убедитесь, что ответ также имеет правильный заголовок, обычно application/json.

- Status Code: Убедитесь, что статус-код соответствует ожидаемому результату.

3.3. Типы тестирования через Postman

* Функциональное тестирование: Проверка, что API работает согласно спецификациям и требованиям.
* Тестирование производительности: Регулярное тестирование, чтобы убедиться в быстродействии API под нагрузкой.
* Тестирование безопасности: Проверка доступа к API, чтобы убедиться, что защищенные ресурсы нельзя получить без необходимости.
* Тесты на валидацию: Проверки на корректность введенных данных и обработку ошибок.
* Регрессионное тестирование: Проверка, что изменения в коде не нарушили существующую функциональность.

3.4. Ожидаемые коды ответа (status codes)

Успешное создание:

201 Created: Успешное создание пользователя.

Ошибка валидации:

400 Bad Request: Неправильные данные в запросе (например, пустое имя, некорректный email).

Попытка создать пользователя с уже существующим email:

409 Conflict: Конфликт, связанный с тем, что email уже используется.

3.5. Оформление тест-кейса

Тест-кейс для проверки API создания пользователя:

Тест-кейс ID: TC001

**Название:** Проверка успешного создания пользователя

**Описание:** Проверка, что новый пользователь создается успешно при наличии действительных данных.

**Предусловия:** API должен быть доступен.

**Шаги:**

1. Отправить POST-запрос на /api/v1/users с телом запроса

{

"name": "Анна",

"email": "[anna@example.com](mailto:anna@example.com)",

"age": 27

}

2. Проверить, что код ответа равен 201 Created.

3. Проверить, что в ответе содержится id, name, email, age, и status равен "active".

**ОР:**

1. Код ответа: 201 Created.

2. Тело ответа соответствует следующим ожидаемым данным:

{

"id": 123, // ID должен быть уникальным

"name": "Анна",

"email": "[anna@example.com](mailto:anna@example.com)",

"age": 27,

"status": "active"

}