Документация к решению тестового задания https://github.com/Leilynne/nevatrip

1. Описание задачи.

Для решения тестового задания необходимо написать функцию, результатом работы которой будет добавление заказа в таблицу.

2. Архитектура и структура базы данных

Для решения задания мне понадобилось две сущности:

Сущность Orders

-id (int) – уникальный идентификатор заказа

-event_id (int) – идентификатор события, к которому привязан заказ

-event_date (datetime) – дата и время события, на которое куплены билеты

-total_price (int) – общая стоимость заказа

-created_at (timestamp) – временная отметка создания записи

-updated_at (timestamp) - временная отметка редактирования записи

-связь Tickets “один ко многим” с сущностью Ticket. Заказ может содержать несколько билетов.

Сущность Tickets

-id (int) – уникальный идентификатор билета

-type (enum) – тип билета. Перечисление TicketType определяет возможные типы билетов (adult, kid, group, benefit)

-price (int) – стоимость билета

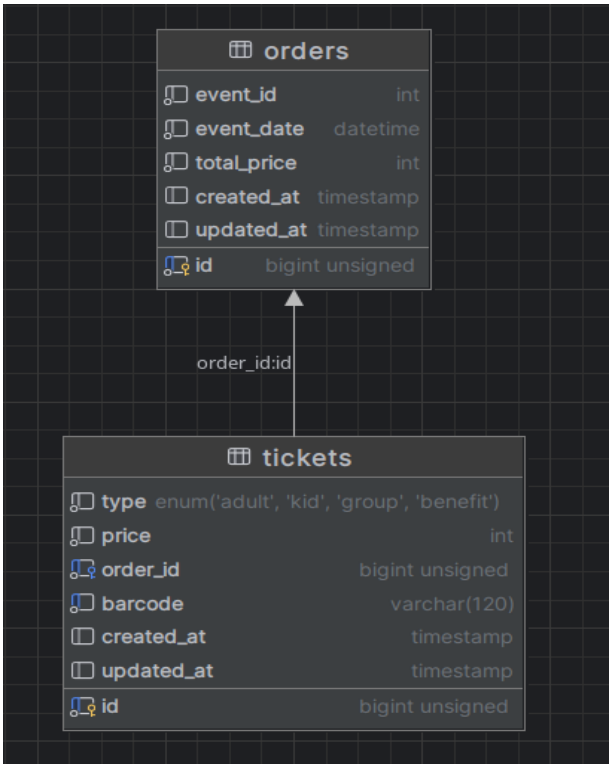
-order_id (int) – идентификатор заказа, к которому привязан билет

-barcode (string) – уникальный код билета

-created_at (timestamp) – временная отметка создания записи

-updated_at (timestamp) - временная отметка редактирования записи

-связь Order “многие к одному” с сущностью Order. Каждый билет принадлежит только одному заказу.



Обоснование решения задания 2.

Согласно заданию 2.1 к основным типам билетам (Adult и Kid) добавляются еще два типа: Group и Benefit. Также указано, что в дальнейшем могут быть добавлены и другие типы билетов.

Для решения задания мною была разработана новая структура базы данных из двух таблиц Orders и Tickets.

Новая структура позволяет обеспечить масштабируемость и гибкость, так как теперь добавление новых типов билетов не потребует изменения таблицы, так как информация о билетах вынесена в отдельную таблицу.

Также новая структура помогает избежать дублирования информации в таблице Orders.

Согласно заданию 2.2 требуется обеспечить возможность проверки каждого билета по отдельности по его уникальному баркоду.

Новая структура нормализует данные, так как нормализация предполагает разделение данных на несколько таблиц так, чтобы каждая таблица содержала только те данные, которые относятся к одной сущности.

В предложенной мной структуре таблица Orders хранит данные только о самом заказе (идентификатор заказа, id события, на которое куплены билеты, дата и время события, а также общая сумма заказа), таблица Tickets хранит информацию о каждом билете (тип билета, его стоимость, количество в заказе билетов конкретного типа, уникальный баркод). Такое разделение улучшает читаемость данных и также упрощает управление базы данных.

Конечный вид таблицы Orders

	id	event_id	event_date	total_price	created_at	updated_at
1	1	5	2024-12-12 00:00:00	2200	2024-11-13 09:14:01	2024-11-13 09:14:01
2	2	5	2024-12-12 00:00:00	2200	2024-11-13 09:15:46	2024-11-13 09:15:46
3	3	5	2024-12-12 12:12:12	2200	2024-11-13 09:24:48	2024-11-13 09:24:48
4	4	5	2024-12-12 12:12:12	2200	2024-11-13 09:26:07	2024-11-13 09:26:07
5	5	5	2024-12-12 12:12:12	2200	2024-11-13 19:08:38	2024-11-13 19:08:38
6	6	5	2024-12-12 12:12:12	2200	2024-11-13 19:09:25	2024-11-13 19:09:25

Конечный вид таблицы Tickets

	id	type	price	order_id	barcode	created_at	updated_at
1	1	adult	500	2	000119121912331222131625140631130111091000220128020 5	2024-11-13 09:15:46	2024-11-13 09:15:46
2	2	adult	500	2	000119121912331222131625140631130111091000220128020 6	2024-11-13 09:15:46	2024-11-13 09:15:46
3	3	kid	400	2	000119121912331222131625140631130111091000220128020 7	2024-11-13 09:15:46	2024-11-13 09:15:46
4	4	kid	400	2	000119121912331222131625140631130111091000220128020 8	2024-11-13 09:15:46	2024-11-13 09:15:46
5	5	kid	400	2	000119121912331222131625140631130111091000220128020 9	2024-11-13 09:15:46	2024-11-13 09:15:46
6	6	adult	500	3	000119121913133425201727202216201012001633013231252 0	2024-11-13 09:24:48	2024-11-13 09:24:48
7	7	adult	500	3	000119121913133425201727202216201012001633013231252 2	2024-11-13 09:24:48	2024-11-13 09:24:48
8	8	kid	400	3	000119121913133425201727202216201012001633013231252 3	2024-11-13 09:24:48	2024-11-13 09:24:48
9	9	kid	400	3	000119121913133425201727202216201012001633013231252 5	2024-11-13 09:24:48	2024-11-13 09:24:48
10	10	kid	400	3	000119121913133425201727202216201012001633013231252 6	2024-11-13 09:24:48	2024-11-13 09:24:48
11	11	adult	500	4	000119121913161108261732082704340115091429021928130 9	2024-11-13 09:26:07	2024-11-13 09:26:07
12	12	adult	500	4	000119121913161108261732082704340115091429021928131 0	2024-11-13 09:26:07	2024-11-13 09:26:07
13	13	kid	400	4	000119121913161108261732082704340115091429021928131 1	2024-11-13 09:26:07	2024-11-13 09:26:07
14	14	kid	400	4	000119121913161108261732082704340115091429021928131 2	2024-11-13 09:26:07	2024-11-13 09:26:07
15	15	kid	400	4	000119121913161108261732082704340115091429021928131 3	2024-11-13 09:26:07	2024-11-13 09:26:07
16	16	adult	500	5	000119122014293513123233271403060301230701193100281 1	2024-11-13 19:08:38	2024-11-13 19:08:38
17	17	adult	500	5	000119122014293513123233271403060301230701193100281 2	2024-11-13 19:08:38	2024-11-13 19:08:38
18	18	kid	400	5	000119122014293513123233271403060301230701193100281 3	2024-11-13 19:08:38	2024-11-13 19:08:38
19	19	kid	400	5	000119122014293513123233271403060301230701193100281 4	2024-11-13 19:08:38	2024-11-13 19:08:38
20	20	kid	400	5	000119122014293513123233271403060301230701193100281 5	2024-11-13 19:08:38	2024-11-13 19:08:38
21	21	adult	500	6	000119122014321320000734083434340206220115130004070 8	2024-11-13 19:09:25	2024-11-13 19:09:25
22	22	adult	500	6	000119122014321320000734083434340206220115130004070 9	2024-11-13 19:09:25	2024-11-13 19:09:25
23	23	kid	400	6	000119122014321320000734083434340206220115130004071 0	2024-11-13 19:09:25	2024-11-13 19:09:25
24	24	kid	400	6	000119122014321320000734083434340206220115130004071 1	2024-11-13 19:09:25	2024-11-13 19:09:25
25	25	kid	400	6	000119122014321320000734083434340206220115130004071 2	2024-11-13 19:09:25	2024-11-13 19:09:25

3. **Описание классов.**

Класс AddOrderRequest

Назначение: валидирует данные заказа, поступающих в OrderController.

Правила валидации:

- event_id – integer, required. При наличии таблицы Events можно было бы проверить id на существование в таблице exists:events,id. Это идентификатор события, на которое куплены билеты.

-event_date – datetime, required. Дата и время события, для которого создается заказ. В ТЗ указан тип varchar 10, но при этом в примере указан формат именно даты и времени.

-ticket_adult_price – integer, required при наличии icket_adult_quantity. Минимальное значение 0. Это стоимость взрослого билета.

-ticket_adult_quantity- integer. Минимальное значение 0. Это количество детских билетов.

-ticket_kid_price – integer, required при наличии icket_kid_quantity. Минимальное значение 0. Это стоимость детского билета.

-ticket_kid_quantity- integer. Минимальное значение 0. Это количество детских билетов.

-ticket_group_price – integer, required при наличии icket_group_quantity. Минимальное значение 0. Это стоимость группового билета.

-ticket_group_quantity- integer. Минимальное значение 0. Это количество групповых билетов.

-ticket_benefit_price – integer, required при наличии icket_benefit_quantity. Минимальное значение 0. Это стоимость льготного билета.

-ticket_benefit_quantity- integer. Минимальное значение 0. Это количество льготных билетов.

Поля со стоимостью билетов и поля с их количеством для каждого типа билетов валидируются вместе. Если передано количество взрослых билетов, то должна быть передана и цена взрослого билета.

Класс OrderAddListCommand

Назначение: класс для передачи параметров запроса в хэндлер в виде четко типизированного объекта.

Параметры конструктора: eventId, eventDate, ticketAdultPrice, ticketAdultQuantity, ticketKidPrice, ticketKidQuantity, ticketGroupPrice, ticketGroupQuantity, ticketBenefitPrice, ticketBenefitQuantity.

Все параметры передаются как публичные свойства для упрощения доступа к данным в обработчике.

Класс OrderController.

Назначение: контроллер, который отвечает за создание заказа на событие с разными типами билетов (взрослые, детские, групповые и льготные). Использует OrderAddListHandler для обработки данных заказа и взаимодействия с внешним API для бронирования билетов.

Метод: **store** (AddOrderRequest $request): OrderResource

Назначение: Метод принимает данные нового заказа, обрабатывает их через OrderAddListHandler и возвращает ресурс заказа в формате OrderResource.

Параметры: $request (AddOrderRequest): Объект запроса, содержащие данные заказа, которые прошли валидацию.

Возвращаемые данные: OrderResource: Объект, представляющий созданный заказ.

Обработка данных:

1. Метод store получает данные из запроса через метод validated() объекта AddOrderRequest, что гарантирует, что все данные успешно прошли валидацию.

2. Данные заказа передаются в команду OrderAddListCommand, где информация о событии и типах билетах, их количество и стоимость, подготавливается для обработки.

3. OrderAddListHandler обрабатывает команду и взаимодействует с внешним API для выполнения бронирования и подтверждения заказа.

4. После успешного создания заказа метод возвращает его данные в формате OrderResource.

Класс OrderAddListHandler

Назначение: хэндлер, который обрабатывает команду OrderAddListCommand, выполняя создание заказа с билетами для события. Он взаимодействует с внешним API для бронирования заказов, генерирует баркоды для билетов, взаимодействует с внешним API для подтверждения заказа, а также сохраняет заказы и билеты в базу данных после успешного бронирования.

Метод: **handle** (OrderAddListCommand $command): OrderDTO

Назначение:

Обрабатывает команду для создания заказа. Хэндлер вычисляет общую стоимость заказа, генерирует баркоды для билетов, выполняет бронирование через внешний API и сохраняет заказ с билетами в базу данных.

Параметры: $command (OrderAddListCommand): Команда, содержащая данные заказа, такие как количество и цены билетов для разных типов (взрослые, детские, групповые и льготные билеты).

Возвращаемое значение: OrderDTO: DTO, представляющее заказ, после того как он был создан и сохранён в базе данных.

Логика:

Извлекает данные из команды (OrderAddListCommand), включая информацию о количестве и цене для различных типов билетов.

- Проверяет, что хотя бы один билет был добавлен в заказ (иначе выбрасывает исключение).
- Вычисляет общую стоимость заказа на основе количества и цен билетов.
- Генерирует баркоды для билетов с помощью метода postBook().
- Отправляет запрос на внешний API для бронирования заказа.
- Отправляет запрос на внешний API для подтверждения брони.
- После успешного бронирования сохраняет заказ в базу данных.
- Для каждого типа билетов (взрослый, детский, групповой, льготный) создаёт записи в таблице билетов.
- Возвращает DTO для созданного заказа через маппер

Может выбрасывать исключение HttpException, если внешний API вернёт ошибку при бронировании или подтверждении.

Метод: **postBook**()

Назначение: Выполняет запрос к внешнему API для бронирования билетов и генерирует баркоды для каждого билета.

Параметры: eventId, eventDate, ticketAdultPrice, ticketAdultQuantity, ticketKidPrice, ticketKidQuantity, ticketGroupPrice, ticketGroupQuantity, ticketBenefitPrice, ticketBenefitQuantity, totalQuantity.

Возвращаемое значение: массив с баркодами для всех купленных билетов.

Логика:

Создаёт уникальные баркоды для каждого билета.

- Выполняет POST-запрос к API для бронирования.
- Если первый запрос не успешен, повторяет его.
- В случае ошибки при бронировании, возвращает выбросить HttpException.

Метод: **createBarcodes** (int $totalQuantity): array

Назначение: генерирует баркод для каждого билета в заказе.

Параметры: totalQuantity – общее количество билетов в заказе, для которых нужно создать баркоды.

Возвращаемое значение: массив строк (баркодов).

Метод: **createTickets**(TicketType $type, int $price, int $orderId, array $barcodes, int $quantity): void

Назначение: Создает записи в базе данных для билетов из заказа.

Параметры: $type, $price, $orderId, $barcodes, $quantity.

Метод: **ulidToNumericString** (Ulid $ulid): string

Назначение: Преобразует ULID в строку числовых значений для создания уникальных баркодов.

Параметры: $ulid

Возвращаемое значение: строка из цифровых символов (баркод)

Логика:

Использует сгенерированный ULID, преобразует каждый буквенный символ в цифровой и добавляя 0 перед цифровым символом, для обеспечения уникальности кода и снижения разночтения символов (02 при отсуствии добавление 0 могло бы читаться как 0 или как 2, что увеличивает вероятность генерации одинакового кода. При добавлении 0 разночтение невозможно, что обеспечивает нужный уровень уникальности кода).

Класс OrderMapper

Назначение: Маппер, предназначенный для преобразования модели Order в OrderDTO.

Метод: **mapModelToDTO**(Order $order): OrderDTO

Параметры: $order - модель заказа, содержащая данные о событии.

Возвращаемое значение: OrderDTO – объект DTO, представляющий заказ.

Класс OrderDTO

Назначение: объект, представляющий данные заказа для передачи между слоями приложения.

Параметры конструктора: id, eventId, eventDate, price, tickets.

Класс OrderResource

Назначение: Задает формат ответа API.

Параметры конструктора: OrderDTO

Класс TicketMapper

Назначение: Маппер, предназначенный для преобразования модели Ticket в TicketDTO.

Метод: **mapModelToDTO**(Ticket $ticket): TicketDTO

Параметры: $ticket – модель билета

Возвращаемое значение: TicketDTO – объект DTO, представляющий билет.

Метод: **mapModelToDTOArray**(Collection $tickets): array

Параметры: $tickets – коллекция моделей Ticket

Возвращаемое значение: array – массив объектов TicketDTO.

Класс TicketResource

Назначение: Задает формат ответа API.

Параметры конструктора: TicketDTO

Класс MockApiServiceProvider

Назначение: сервис-провайдер, который конфигурирует моки для внешних API запросов, позволяя имитировать ответы API.

Метод: **boot**(): void

Логика:

- Логинит пользователя с конкретным id с целью тестирования.
- Настраивает моки на запросы, отправляемые через Http фасад, перехватывая запросы к https://api.site.com/book и https://api.site.com/approve.

Возвращаемое значение: возвращает случайный ответ.

API Документация для Заказов

Эндпоинт: Создание Заказа

URL: /api/orders

Метод: POST

Описание: создает новый заказ на билеты для различных категорий с указанием их цен и количества.

Заголовки запроса:

- Content-Type: application/json
- Accept: application/json

Тело Запроса

Поле	Тип	Обязательное	Описание
event_id	int	Да	Уникальный идентификатор события.
event_date	string	Да	Дата и время события в формате Y-m-d H:i:s.
ticket_adult_price	int	Обязательно при наличии ticket_adult_quantity	Цена взрослого билета. Обязателен, если указано количество взрослых билетов.
ticket_adult_quantity	int	Нет	Количество взрослых билетов, минимум 0.
ticket_kid_price	int	Обязательно при наличии ticket_kid_quantity	Цена детского билета. Обязателен, если указано количество детских билетов.
ticket_kid_quantity	int	Нет	Количество детских билетов, минимум 0.
ticket_group_price	int	Обязательно при наличии ticket_group_quantity	Цена группового билета. Обязателен, если указано количество групповых билетов.
ticket_group_quantity	int	Нет	Количество групповых билетов, минимум 0.
ticket_benefit_price	int	Обязательно при наличии ticket_benefit_quantity	Цена льготного билета. Обязателен, если указано количество льготных билетов.
ticket_benefit_quantity	int	Нет	Количество льготных билетов, минимум 0.

Пример Запроса


```
{
  "event_id": 123,
  "event_date": "2024-11-13 19:00:00",
  "ticket_adult_price": 100,
  "ticket_adult_quantity": 2,
  "ticket_kid_price": 50,
  "ticket_kid_quantity": 1,
  "ticket_group_price": 80,
  "ticket_group_quantity": 3,
  "ticket_benefit_price": 30,
  "ticket_benefit_quantity": 1
}
```


Ответ:

Успех (201 Created)

Если заказ успешно создан, возвращается статус 201 Created с данными созданного заказа.


```
{
"data": {
"id": 6,
"event_id": 5,
"event_date": "2024-12-12T12:12:12.000000Z",
"price": 2200,
"tickets": [
{
"id": 21,
"type": "Взрослый",
"price": 500,
"barcode": "00011912201432132000073408343434020622011513000407 08"
},
{
"id": 22,
"type": "Взрослый",
"price": 500,
```

"barcode": "000119122014321320000734083434340206220115130004070 9"
},
{
"id": 23,
"type": "Детский",
"price": 400,
"barcode": "000119122014321320000734083434340206220115130004071 0"
},
{
"id": 24,
"type": "Детский",
"price": 400,
"barcode": "000119122014321320000734083434340206220115130004071 1"
},
{
"id": 25,
"type": "Детский",
"price": 400,
"barcode": "000119122014321320000734083434340206220115130004071 2"
}]
}
}

Ошибки:

422 Unprocessable Entity: Ошибка валидации, как правило, из-за отсутствующих обязательных полей или некорректных значений.

Пример ответа с ошибкой:

{
"message": "Данные неверны.",
"errors": {
"event_id": [
"Поле event_id обязательно."
],
"ticket_adult_price": [

```
      "Поле ticket_adult_price обязательно при наличии ticket_adult_quantity."
    ]
  }
}
```


422 Unprocessable Entity: Ошибка работы с внешним API

Пример ответа с ошибкой:


```
{
  "message": "event cancelled"
}
```