C2 - Phase one - PHP API Development

Competitor Information

The final website has to be available at `http://<hostname>/module-b/phase1`. For example, the concerts endpoint is then reachable at `http://<hostname>/module-b/phase1/api/v1/concerts`.

Tests are provided that check for correct implementation of the following requirements.

To run the tests, execute `composer test`. If the API is available under the correct URL, the tests should automatically reach it. If for development, the API is available at a different URL, it can be overwritten with the environment variable `BASE_URL` (e.g. `BASE_URL=http://localhost:8080/ composer test`). Do *not* include `/api/v1` in the path in this case.

To safe time and only run a subset of tests, it is possible to filter tests:

- by their filename: `composer test -- tests/ConcertListingTest.php`
- by their test name: `composer test -- --filter testConcertListingAllStrictEquals`

Please note that when running the tests, the database gets reset to ensure having exactly the same state for every test run. So your changes to the database will be lost.

Database design

A database dump is already provided to you. Import it into your database and use the provided schema and data for developing the application.

The database has to be named `concerts` and the MySQL user `competitor` needs all privileges on this database.

API

To allow parallel development of a mobile app, you are provided an API specification and it is therefore important that it is exactly followed. Provided tests will verify a correct implementation.

> **General information:**
>
> - The response body contains some static example data. Dynamic data from the database should be used.
> - Placeholder parameters in the URL are marked with curly braces (e.g. `{slug}`).
> - The order of properties in objects does not matter, but the order in an array does.
> - If not specified otherwise, the `Content-Type` header of a response is always `application/json`

`GET /api/v1/concerts`

Returns a list of all available concerts and all their performances.

Concerts are sorted by their artist name in ascending order. Shows are sorted by their date while the earliest show should be returned first.

Responses

Status Code	Data

200	**Description**: Returned on a successful request.
	
	Body
	
	<pre>{
	  "concerts": [
	  {
	  "id": 1,
	  "artist": "Opus",
	  "location": {
	  "id": 1,
	  "name": "Oper Graz"
	  },
	  "shows": [
	  {
	  "id": 1,
	  "start": "2021-10-02T20:00:00Z",
	  "end": "2021-10-02T23:00:00Z"
	  }
	  ]
	  }
	  ]
	}</pre>

`GET /api/v1/concerts/{concert-id}`

Returns a single concert.

Shows are sorted by their date while the earliest show should be returned first.

Request

Path parameters

Parameter	Type	Description
`concert-id`	int	Database ID of the concert.

Responses

Status Code	Data
200	**Description**: Returned on a successful request. **Body** `{ "concert": { "id": 1, "artist": "Opus", "location": { "id": 1, "name": "Oper Graz" }, "shows": [ { "id": 1, "start": "2021-10-02T20:00:00Z", "end": "2021-10-02T23:00:00Z" } ] } }`
404	**Description**: Returned if a concert with the specified ID does not exist. **Body** `{ "error: "A concert with this ID does not exist" }`

`GET /api/v1/concerts/{concert-id}/shows/{show-id}/seating`

Returns seating information for the given concert and contains all rows and their capacity. In each row, seats are numbered from 1 to the total number of seats per row.

For each row, all unavailable seats (their `number` field) are also returned.

Rows are sorted by their `order` and seats by their `number` field in ascending order.

Request

Path parameters

Parameter	Type	Description
`concert-id`	int	Database ID of the concert.
`show-id`	int	Database ID of a specific show from the concert.

Responses

Status Code	Data
200	**Description**: Returned on a successful request. **Body** `{ "rows": [ { "id": 1, "name": "Floor 1", "seats": { "total": 50, "unavailable": [1, 2, 15, 16, 17] } } ] }`
404	**Description**: Returned if a concert or show with the specified ID does not exist or the show does not belong to the specified concert. **Body** `{ "error: "A concert or show with this ID does not exist" }`

`POST /api/v1/concerts/{concert-id}/shows/{show-id}/reservation`

Reserves one or more seats so they cannot be booked or reserved by someone else. In a later request, the reservation can be promoted to a full ticket or replaced with another one. If no subsequent request is performed, the reservation will eventually time out (see request body) and is free again to be reserved or booked by someone else.

Each reservation will return a reservation token. With this token, the reservation can later be replaced or promoted to a full ticket. It serves as a way to authenticate a user in subsequent requests without the need for a login.

The generation of the token is not specified and implementation is up to the developer. The only requirement is that it cannot easily be guessed and so should be random in any form.

Request

Path parameters

Parameter	Type	Description
`concert-id`	int	Database ID of the concert.
`show-id`	int	Database ID of a specific show from the concert.

Header

```
Content-Type: application/json
```

Body

```
{
    "reservation_token": "...",
    "reservations": [
        {
            "row": 1,
            "seat": 15
        },
        {
            "row": 1,
            "seat": 16
        }
    ],
    "duration": 300
}
```

The `reservation_token` is **optional** and, if specified, is replaced with the reservation token returned by a previous reservation request. If a token is specified, the previous reservation for the same show, if it exists, is deleted and the same token is returned in the response (no new token gets generated). If there is no token specified, a new one will get randomly generated and returned.

Multiple seats can be reserved. Both, `row` and `seats` are **required** for each reserved seat. `row` relates to the `id` field of the row and `seat` to the `number` of the seat within the row.

If an empty array is specified for `reservations`, the previously reserved seats from the reservation token are deleted but no new seats are reserved.

A duration can be specified, defining how long the seats are reserved in **seconds**. A duration is **optional** and if not specified, it will default to **5 minutes**. If a duration is given, it **must be at least 1 second** and cannot **not be longer than 5 minutes**.

Responses

Status Code	Data
201	**Description**: Returned on a successful request. `reserved_until` specifies the date and time until the seats are reserved in the ISO 8601 format. `reservation_token` is either the reservation token specified in the request body or a newly generated one. **Body** `{ "reserved": true, "reservation_token": "...", "reserved_until": "2021-09-24T10:25:46Z" }`
403	**Description**: Returned if the reservation token is not valid (it was not created before). Is only returned if a reservation token is specified in the request body. **Body** `{ "error: "Invalid reservation token" }`
404	**Description**: Returned if a concert or show with the specified ID does not exist or the show does not belong to the specified concert. **Body** `{ "error: "A concert or show with this ID does not exist" }`

422	**Description**: Returned if validation failed. Only fields for which the validation failed should be included in the response. If an error message can be applied to multiple seats, only the first one from the request body that it can be applied to is specified in the validation message.
	
	The following validation messages are possible:
	
	• The {field} field is required.
	  ◦ `{field}` is replaced with the field name, e.g. row
	• Seat {seat} in row {row} is invalid.
	  ◦ `{seat}` is replaced with the seat `number` and `{row}` is replaced with the row `id`.
	  ◦ Returned if one of the row IDs does not exist, does not belong to the given show, or one of the seats is out of bounds for the given row.
	• Seat {seat} in row {row} is already taken.
	  ◦ `{seat}` is replaced with the seat `number` and `{row}` is replaced with the row `id`.
	• The duration must be between 1 and 300.

Body

```
{
    "error: "Validation failed",
    "fields": {
        "reservations": "Seat 15 in row 1 is already taken.",
        "duration": "The duration must be between 1 and 300."
    }
}
```

`POST /api/v1/concerts/{concert-id}/shows/{show-id}/booking`

Upgrades a reservation to a full ticket. If the reservation contains multiple seats, one ticket per seat will get created. Only reservations for the specified show are upgraded.

All created tickets will be assigned to the same booking so they can later be easily resolved.

Tickets are sorted by their row (`order` field) and seat (`number` field) in ascending order.

Request

Path parameters

Parameter	Type	Description
`concert-id`	int	Database ID of the concert.
`show-id`	int	Database ID of a specific show from the concert.

Header

```
Content-Type: application/json
```

Body

```
{
    "reservation_token": "...",
    "name": "John Doe",
    "address": "Bahnhofstrasse 15",
    "city": "Graz",
    "zip": "8010",
    "country": "Austria"
}
```

`reservation_token` is **required** and contains the token returned by the reservation endpoint. `name` , `address` , `city` , `zip` and `country` are all **required**, of type **string**, and later used for creating an invoice.

Responses

Status Code	Data
201	**Description**: Returned if the reservation was successfully updated to one or more full tickets. The ticket `code` is a 10 character long uppercase alphanumeric string that is generated randomly for each ticket. It will be presented during entrance at the concert and used to authenticate on further requests.

Body


```
{
    "tickets": [
        {
            "id": 1,
            "code": "QVLJTWK4Y7",
            "name": "John Doe",
            "created_at": "2021-09-24T11:02:20Z",
            "row": {
                "id": 1,
                "name": "Floor 1"
            },
            "seat": 35,
            "show": {
                "id": 1,
                "start": "2021-10-02T20:00:00Z",
                "end": "2021-10-02T23:00:00Z",
                "concert": {
                    "id": 1,
                    "artist": "Opus",
                    "location": {
                        "id": 1,
                        "name": "Oper Graz"
                    }
                }
            }
        }
    ]
}
```


401

Description: Returned if the provided reservation token is invalid.

Body


```
{
    "error: "Unauthorized"
}
```


404

Description: Returned if a concert or show with the specified ID does not exist or the show does not belong to the specified concert.

Body


```
{
    "error: "A concert or show with this ID does not exist"
}
```


422

Description: Returned if validation failed. Only fields for which the validation failed should be included in the response.

The following validation messages are possible:

- The {field} field is required.
- The {field} must be a string.

`{field}` is replaced with the field name.

Body

```
{
    "error: "Validation failed",
    "fields": {
        "reservation_token": "The reservation token field is required.",
        "name": "The name field is required.",
        "city": "The city must be a string.",
        "zip": "The zip must be a string."
    }
}
```

`POST /api/v1/tickets`

Returns all tickets of the specified booking.

Users can specify their name and one of the ticket codes. If the name is correct for the ticket, all tickets from the same booking are returned.

Tickets are sorted by their row (`order` field) and seat (`number` field) in ascending order.

Request

Header

```
Content-Type: application/json
```

Body

```
{
    "code": "QVLJTWK4Y7",
    "name": "John Doe"
}
```

Responses

Status Code	Data
200	**Description**: Returned on a successful request. **Body** `{ "tickets": [{ "id": 1, "code": "QVLJTWK4Y7", "name": "John Doe", "created_at": "2021-09-24T11:02:20Z", "row": { "id": 1, "name": "Floor 1" }, "seat": 35, "show": { "id": 1, "start": "2021-10-02T20:00:00Z", "end": "2021-10-02T23:00:00Z", "concert": { "id": 1, "artist": "Opus", "location": { "id": 1, "name": "Oper Graz" } } } }] }`
401	**Description**: Returned if either `code` or `name` is missing or no ticket matches.

	Body
	
	{
	"error: "Unauthorized"
	}

`POST /api/v1/tickets/{ticket-id}/cancel`

Deletes the specified ticket. The seat assigned to this ticket is available again.

All other tickets from the assigned booking remain active.

Request

```
Content-Type: application/json
```

Body

```
{
    "code": "QVLJTWK4Y7",
    "name": "John Doe"
}
```

Responses

Status Code	Data
204	**Description**: Returned if the ticket was successfully deleted. **Body**

401	**Description**: Returned if either `code` or `name` is missing or they do not match the ticket.
	
	Body
	
	{
	"error: "Unauthorized"
	}

404	**Description**: Returned if a ticket with the specified ID does not exist.
	
	Body
	
	{
	"error: "A ticket with this ID does not exist"
	}