POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

EICTA - 3rd Project Delivery - MongoDB & Web

Group 10:

Luigi Tiberio [redacted]
Stefano Moreschi [redacted]
Mattia Silvestri [redacted]

Academic Year: 2025/2026

Contents

1 | MongoDB

1.1. Document Structure and Modeling Choices

1.1.1. Document Structure and Containment Hierarchy

In terms of attribute definition, each Festival document includes the following typed fields:

- **festival_name** (String): the name of the event.
- **festival_start_date** and **festival_end_date** (String, ISO 8601): the temporal interval of the festival.
- **festival_location** (String): the hosting city.
- **stands** (Array of Stand sub-documents): the set of stands participating in the festival.
- **tickets** (Array of Ticket sub-documents): the ticket types available for the event.

Each Stand sub-document contains:

- **stand_name** (String): the name of the stand.
- **stand_sponsor** (String): the associated company or sponsor.
- **product_drink**, **product_food**, **product_merchandise** (Arrays of Product sub-documents): the categories of products sold by the stand.

All product sub-documents share a common nested structure:

- **product.product_name** (String): the product's name.
- **product.product_price** (Number): the unit price.

Drink products additionally include:

- **drink_exp_date** (String, ISO 8601): expiration date.

Food products include:

- **food_exp_date** (String, ISO 8601): expiration date.
- **food_type** (String): type of food (e.g., vegan, vegetarian, meat-based).

Merchandise products introduce no additional attributes beyond the shared product structure.

Each Ticket sub-document contains:

- **ticket_id** (String): identifier of the ticket.
- **ticket_type** (String): category of the ticket.
- **ticket_price** (Number): price.
- **ticket_validity_start** and **ticket_validity_end** (String, ISO 8601): validity range.

The MongoDB database is organized around a single collection in which each document represents one festival. This design choice follows the relational ER model, where the Festival entity acts as the central element from which all other entities depend. In MongoDB, this translates naturally into a hierarchical structure in which the dependent entities, Stands, Products, and Tickets, are embedded inside their parent Festival document.

At the top level, each Festival document contains descriptive attributes such as its name, start and end dates, and location. It also embeds two main arrays: one containing all stands associated with the festival, and one listing all ticket types. This approach preserves the original one-to-many relationships while benefiting from MongoDB's document-oriented design, where related data is stored together and retrieved without join-like operations.

Each stand is modeled as a sub-document inside the festival. A stand contains its descriptive attributes as well as three arrays representing the product categories (drink, food, merchandise). These arrays correspond to the ISA hierarchy defined in the conceptual model: a Product super-entity with Drink, Food, and Merchandise specializations. In MongoDB, the shared attributes of the Product super-entity are placed inside the nested product sub-document, while subtype-specific attributes appear only where relevant. This strategy preserves the conceptual hierarchy without requiring multiple collections or inter-document references.

The final embedded component of the Festival document is the tickets array. Each ticket is represented by a sub-document containing its type, price, and validity interval. Since tickets belong exclusively to a single festival and have no external interactions, embedding

them maintains semantic consistency while improving data locality.

1.1.2. Database Considerations

The design of the MongoDB schema aims to balance fidelity to the conceptual model previously developed in SQL and Neo4j with the strengths of a document-oriented database. Embedding was chosen over referencing whenever an entity is fully dependent on the Festival and does not possess an independent lifecycle. This choice prevents fragmentation across multiple collections, keeps related information physically grouped, and naturally reflects the 1–N relationships present in the ER model.

Another design consideration concerns the representation of temporal attributes. All dates—including festival dates, ticket validity intervals, and product expiration dates—are stored as ISO 8601 strings (`"YYYY-MM-DD"`). Although MongoDB provides native date types, the ISO format preserves chronological order under lexicographical comparison and fully satisfies the requirements of the assignment. Comparison operators such as `$gt`, `$lt` and `$lte` work correctly with ISO strings, allowing date-based queries without conversion.

During the transition from the relational dataset to the MongoDB schema, a key constraint of the assignment required attention: every array of sub-documents must contain at least two elements, and empty arrays are not allowed. In the original dataset, some festivals included only one stand, and several stands contained a single product in one or more categories. To comply with the constraint while avoiding the creation of artificial data, a selective cleaning process was applied. Festivals with only one stand were removed; stands with any product category containing a single element were discarded; and any festival reduced to fewer than two stands after this operation was also removed. The final dataset therefore forms a coherent, self-consistent subset of the original data and satisfies both the conceptual model and the structural requirements imposed by the MongoDB assignment.

1.2. Queries

1.2.1. Creation/Update/Delete queries

Q CREATE 1– Insert a New Festival Document

```
db.festival.insertOne({
  festival_name: "River Lights",
  festival_start_date: "2026-07-10",
  festival_end_date: "2026-07-12",
  festival_location: "Florence",
  stands: [
    {
      stand_name: "Cool Drinks",
      stand_sponsor: "FreshCo Beverages",
      product_drink: [
        {
          product: {
            product_name: "Mango Smoothie 300ml",
            product_price: 3.2
          },
          drink_exp_date: "2030-12-31"
        },
        {
          product: {
            product_name: "Strawberry Lemonade 300ml",
            product_price: 3.7
          },
          drink_exp_date: "2030-12-31"
        }
      ],
      product_food: [
        {
          product: {
            product_name: "Veggie Burger",
            product_price: 7.5
          },
          food_exp_date: "2029-06-01",
          food_type: "vegan"
        },
        {
          product: {
            product_name: "Grilled Halloumi Wrap",
            product_price: 8.0
          },
          food_exp_date: "2029-06-01",
          food_type: "vegetarian"
        }
      ],
      product_merchandise: [
        {
          product: {
            product_name: "Festival T-Shirt",
            product_price: 20.0
          }
        },
        {
          product: {
            product_name: "Reusable Water Bottle",
            product_price: 12.0
          }
        }
      ]
    },
    {
      stand_name: "Tasty Bites",
      stand_sponsor: "Florence Street Foods",
      product_drink: [
        {
          product: {
            product_name: "Iced Tea 400ml",
            product_price: 2.8
          },
          drink_exp_date: "2030-10-01"
        },
        {
          product: {
            product_name: "Sparkling Water 330ml",
            product_price: 2.2
          },
          drink_exp_date: "2030-10-01"
        }
      ],
      product_food: [
        {
          product: {
            product_name: "Margherita Slice",
            product_price: 4.5
          },
          food_exp_date: "2029-07-01",
          food_type: "vegetarian"
        },
        {
          product: {
            product_name: "Chicken Focaccia",
            product_price: 6.5
          },
          food_exp_date: "2029-07-01",
          food_type: "meat"
        }
      ]
    }
  ],
```

```
  tickets: [
    {
      ticket_id: "TCK9001",
      ticket_type: "Standard",
      ticket_price: 35.0,
      ticket_validity_start: "2026-07-10",
      ticket_validity_end: "2026-07-12"
    },
    {
      ticket_id: "TCK9002",
      ticket_type: "VIP",
      ticket_price: 70.0,
      ticket_validity_start: "2026-07-10",
      ticket_validity_end: "2026-07-12"
    }
  ]
});
```

Query Description:

This query inserts a new festival into the festival collection. The document includes the festival's basic information, two stand containing drink, food, and merchandise arrays (each with at least two element), and two tickets. It demonstrates how to create a fully structured MongoDB document that respects the required hierarchy.

Query Result:

```
{
  acknowledged: true,
  insertedId: ObjectId('6935a1f34641bc01a5832ea8')
}
```

Q UPDATE 2 - Add a New Drink to an Existing Stand of an Original Festival

```
db.festival.updateOne(
  {
    festival_name: "Sunset Echo",
    festival_location: "Tokyo",
    festival_start_date: "2024-04-21",
    "stands.stand_name": "Sonic Snack "
  },
  {
    $push: {
      "stands.$.product_drink": {
        product: {
          product_name: "Ginger Spark Soda 330ml",
          product_price: 3.6
        },
        drink_exp_date: "2031-03-15"
      }
    }
  }
);
```

Query Description:

This query updates the original Sunset Echo festival (Tokyo, 2024-04-21) by adding a new drink to the product_drink array of the Sonic Snack stand, using $push with the positional operator $.

Query Result:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Q UPDATE 3 - Update the Price of a Day-Pass Ticket for the "Electric Valley" Festival in Madrid

```
db.festival.updateOne(
  {
    festival_name: "Electric Valley",
    festival_location: "Madrid",
    festival_start_date: "2024-07-01",
    "tickets.ticket_id": "TCK1313"
  },
  {
    $set: {
      "tickets.$.ticket_price": 12
    }
  }
);
```

Query Description:

This query updates the price of the DAY-PASS ticket with ticket_id = "TCK1313" for the Electric Valley festival in Madrid (2024-07-01), using updateOne with $set and the positional operator $ on the tickets array.

Query Result:

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

Q DELETE 4- Delete an Entire Festival Document

```
db.festival.deleteOne(
  {
    festival_name: "Aurora Soundwave",
    festival_location: "Milan",
    festival_start_date: "2024-03-07"
  }
);
```

Query Description:

This query deletes the entire festival document for "Aurora Soundware", held in Milan starting on 2024-03-07, including all nested stands and tickets. The filter includes the three identifying fields required to uniquely match a festival.

Query Result:

```
{
  acknowledged: true,
  deletedCount: 1
}
```

Q DELETE 5- Delete All Festivals Starting Before a date

```
db.festival.deleteMany(
  {
    festival_start_date: { $lt: "2024-05-25" }
  }
);
```

Query Description:

This query deletes all festival documents whose festival_start_date occurs before 2024-05-25. It uses deleteMany() with a $lt (less-than) comparison on the date field.

Query Result:

```
{
  acknowledged: true,
  deletedCount: 4
}
```

1.2.2. Queries with projections and logical query operators

Q 1- Find Italian Festivals in 2025 with Projection and Logical Operators

```
db.festival.find(
  {
    $and: [
      {
        festival_start_date: {
          $gte: "2025-01-01",
          $lte: "2025-12-31"
        }
      },
      {
        $or: [
          { festival_location: "Milan" },
          { festival_location: "Rome" },
          { festival_location: "Naples" }
        ]
      }
    ]
  },
  {
    _id: 0,
    festival_name: 1,
    festival_location: 1,
    festival_start_date: 1
  }
);
```

Query Description:

This query retrieves all festivals that:

- take place in Italy (Milan, Rome, or Naples),
- and start in 2025.

It uses logical operators `$and` and `$or`, together with a projection that returns only `festival_name`, `festival_location`, and `festival_start_date` (excluding `_id`).

Query Result:

```
{
  festival_name: 'Aurora Soundwave',
  festival_start_date: '2025-09-10',
  festival_location: 'Milan'
}
{
  festival_name: 'Sunset Echo',
  festival_start_date: '2025-05-28',
  festival_location: 'Naples'
}
{
  festival_name: 'Velvet Noise',
  festival_start_date: '2025-11-09',
  festival_location: 'Naples'
}
```

Q 2- Stands selling Diet Cola and Pizza Slice

```
db.festival.aggregate([
  {
    $match: {
      stands: {
        $elemMatch: {
          "product_drink.product.product_name": "Diet Cola 330ml",
          "product_food.product.product_name": "Pizza Slice"
        }
      }
    }
  },
  {
    $unwind: "$stands"
  },
  {
    $match: {
      $and: [
        { "stands.product_drink.product.product_name": "Diet Cola 330ml" },
        { "stands.product_food.product.product_name": "Pizza Slice" }
      ]
    }
  },
  {
    $project: {
      _id: 0,
      festival_name: 1,
      festival_location: 1,
      festival_start_date: 1,
      stand_name: "$stands.stand_name"
    }
  }
]);
```

Query Description:

This query returns the name, location, start date of festivals and the names of stands where both “Diet Cola 330ml” and “Pizza Slice” are sold at the same stand.

Query Result:

```
{
  festival_name: 'Sunset Echo',
  festival_start_date: '2024-04-21',
  festival_location: 'Tokyo',
  stand_name: 'Groove & Brew'
}
{
  festival_name: 'Aurora Soundwave',
  festival_start_date: '2024-03-07',
  festival_location: 'Milan',
  stand_name: 'StageFuel'
}
{
  festival_name: 'Midnight Grove',
  festival_start_date: '2024-02-23',
  festival_location: 'Berlin',
  stand_name: 'BeatStreet Stand'
}
{
  festival_name: 'Northern Lights',
  festival_start_date: '2025-08-13',
  festival_location: 'Monaco',
  stand_name: 'Vibe & Sip'
}
{
  festival_name: 'Electric Valley',
  festival_start_date: '2024-07-01',
  festival_location: 'Madrid',
  stand_name: 'Bassline Bites'
}
```

1.2.3. Queries with limit, sort and comparison query operators

Q 3-Find the Next 3 Festivals After a Given Date

```
db.festival.find(
  {
    festival_start_date: { $gt: "2024-02-01" }
  },
  {
    _id: 0,
    festival_name: 1,
    festival_location: 1,
    festival_start_date: 1
  }
)
.sort({ festival_start_date: 1 })
.limit(3);
```

Query Description:

This query retrieves the first 3 festivals that start after 2025-02-01, ordered by festival_start_date in ascending order. It uses a comparison operator ($gt), sorting (sort) and limiting (limit), plus a simple projection

Query Result:

```
{
  festival_name: 'Rhythm Horizon',
  festival_start_date: '2024-02-11',
  festival_location: 'Rome'
}
{
  festival_name: 'Midnight Grove',
  festival_start_date: '2024-02-23',
  festival_location: 'Berlin'
}
{
  festival_name: 'Aurora Soundwave',
  festival_start_date: '2024-03-07',
  festival_location: 'Milan'
}
```

Q4 - Top 3 Stands With the Highest Number of Products

```
db.festival.aggregate([
  {
    $unwind: "$stands"
  },
  {
    $project: {
      _id: 0,
      stand_name: "$stands.stand_name",
      festival_name: 1,
      festival_location: 1,
      festival_start_date: 1,
      total_products: {
        $add: [
          { $size: { $ifNull: [ "$stands.product_drink", [] ] } },
          { $size: { $ifNull: [ "$stands.product_food", [] ] } },
          { $size: { $ifNull: [ "$stands.product_merchandise", [] ] } }
        ]
      }
    }
  },
  {
    $sort: { total_products: -1 }
  },
  {
    $limit: 3
  }
]);
```

Query Description:

This query returns the three stands (across all festivals) that sell the largest total number of products (drinks + food + merchandise). It uses $size and $add to compute the total, then sorts by this value and limits the result to 3.

Query Result:

```
{
  festival_name: 'Northern Lights',
  festival_start_date: '2024-09-24',
  festival_location: 'Rome',
  stand_name: 'Merch & Munch Spot',
  total_products: 16
}
{
  festival_name: 'Northern Lights',
  festival_start_date: '2024-09-24',
  festival_location: 'Rome',
  stand_name: 'Echo Merch & Munch',
  total_products: 16
}
{
  festival_name: 'Midnight Grove',
  festival_start_date: '2024-02-23',
  festival_location: 'Berlin',
  stand_name: 'BeatStreet Stand',
  total_products: 16
}
```

1.2.4. Query including element query operators

Q5 - Retrieve the most recent festival whose festival_start_date field exists.

```
db.festival.find(
  {
    festival_start_date: { $exists: true }
  },
  {
    _id: 0,
    festival_name: 1,
    festival_start_date: 1,
    festival_location: 1
  }
).sort({ festival_start_date: -1 }).limit(1);
```

Query Description:

This query retrieves the most recent festival whose festival_start_date field exists. Uses the $exists element operator and returns only the festival name and start date.

Query Result:

```
{
  festival_name: 'Velvet Noise',
  festival_start_date: '2025-11-09',
  festival_location: 'Naples'
}
```

1.2.5. Query using the group operator

Q6 - Group festivals by location

```
db.festival.aggregate([
  {
    $group: {
      _id: "$festival_location",
      total_festivals: { $sum: 1 }
    }
  }
]);
```

Query Description:

Retrieve how many festivals are held in each location. The query uses the $group aggregation operator to group documents by festival_location and count festivals per location.

Query Result:

```
{
  _id: 'Milan',
  total_festivals: 2
}
{
  _id: 'Barcellona',
  total_festivals: 2
}
{
  _id: 'Philadelphia',
  total_festivals: 1
}
{
  _id: 'Tokyo',
  total_festivals: 2
}
{
  _id: 'Berlin',
  total_festivals: 1
}
```

```
{
  _id: 'Naples',
  total_festivals: 3
}
{
  _id: 'Rome',
  total_festivals: 2
}
{
  _id: 'Madrid',
  total_festivals: 1
}
{
  _id: 'Monaco',
  total_festivals: 2
}
```


1.2.6. Query with an unwind and a group by

Q7- Count tickets per ticket type

```
db.festival.aggregate([
  {
    $unwind: "$tickets"
  },
  {
    $group: {
      _id: "$tickets.ticket_type",
      total_tickets: { $sum: 1 }
    }
  }
]);
```

Query Description:

Unwind the tickets array to consider each ticket individually, then group by ticket_type to count how many tickets of each type exist across all festivals.

Query Result:

```
{
  _id: 'VIP',
  total_tickets: 78
}
{
  _id: 'DAY-PASS',
  total_tickets: 119
}
{
  _id: 'REGULAR',
  total_tickets: 118
}
```

1.2.7. Query with an unwind, a group by, and followed by a filter

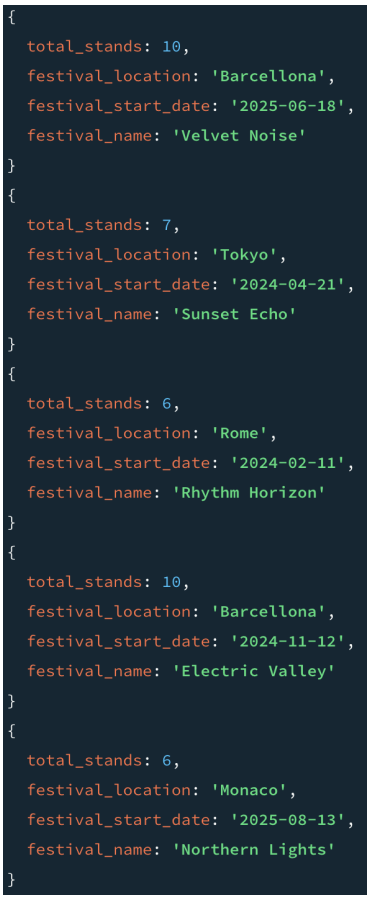
Q8 - Count festivals with more than 5 stands

```
db.festival.aggregate([
  {
    $unwind: "$stands"
  },
  {
    $group: {
      _id: "$festival_name",
      total_stands: { $sum: 1 },
      festival_location: { $first: "$festival_location" },
      festival_start_date: { $first: "$festival_start_date" }
    }
  },
  {
    $match: {
      total_stands: { $gt: 5 }
    }
  },
  {
    $project: {
      _id: 0,
      festival_name: "$_id",
      festival_location: 1,
      festival_start_date: 1,
      total_stands: 1
    }
  }
]);
```

Query Description:

Unwind the stands array, group by festival name to count stand occurrences, and return only festivals with more than 5 stands

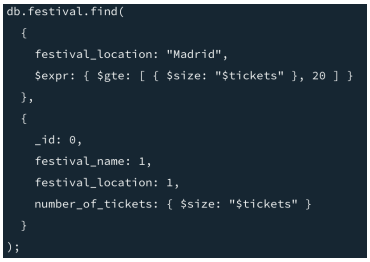
Query Result:



```
{
  total_stands: 10,
  festival_location: 'Barcellona',
  festival_start_date: '2025-06-18',
  festival_name: 'Velvet Noise'
}
{
  total_stands: 7,
  festival_location: 'Tokyo',
  festival_start_date: '2024-04-21',
  festival_name: 'Sunset Echo'
}
{
  total_stands: 6,
  festival_location: 'Rome',
  festival_start_date: '2024-02-11',
  festival_name: 'Rhythm Horizon'
}
{
  total_stands: 10,
  festival_location: 'Barcellona',
  festival_start_date: '2024-11-12',
  festival_name: 'Electric Valley'
}
{
  total_stands: 6,
  festival_location: 'Monaco',
  festival_start_date: '2025-08-13',
  festival_name: 'Northern Lights'
}
```

1.2.8. Query with two conditions evaluated separately on the sub-documents of an array

Q9 - Festivals located in Madrid and having at least 20 tickets



```
db.festival.find(
  {
    festival_location: "Madrid",
    $expr: { $gte: [ { $size: "$tickets" }, 20 ] }
  },
  {
    _id: 0,
    festival_name: 1,
    festival_location: 1,
    number_of_tickets: { $size: "$tickets" }
  }
);
```

Query Description:

Return all festivals that (1) take place in Madrid, and (2) have at least 20 ticket entries in the `tickets` array.

The conditions are evaluated separately:

- one condition concerns a field of the main document (`festival_location`)

- one condition concerns an array (`tickets`)

Query Result:

```
{
  festival_name: 'Electric Valley',
  festival_location: 'Madrid',
  tickets_count: 22
}
```

1.2.9. Query with two conditions evaluated simultaneously on the sub documents of an array

Q10 - Festivals with a specific stand name and sponsor

```
db.festival.find(
  {
    stands: {
      $elemMatch: {
        stand_name: "Harmony Hut",
        stand_sponsor: "EchoStream Media"
      }
    }
  },
  {
    _id: 0,
    festival_name: 1,
    festival_location: 1,
    festival_start_date: 1
  }
);
```

Query Description:

Return all festivals that have at least one stand named "Harmony Hut" and sponsored by "EchoStream Media". The two conditions (stand_name and stand_sponsor) are evaluated simultaneously on the same stand sub-document of the stands array using $elemMatch. The query returns the festival name, location, and start date.

Query Result:

```
{
  festival_name: 'Electric Valley',
  festival_start_date: '2024-01-16',
  festival_location: 'Monaco'
}
```

2 | Web

2.1. Promotional web page for Euphoric Events

2.1.1. Description

Our web page for *Electric Valley 2026* is a promotional single-page site designed to present a three-day electronic music festival organized by Euphoric Events. The structure is divided into clear thematic sections (Hero, About, Line-up, Stages, Food & Drinks, Tickets, Performances, Location, FAQ) to guide visitors through the main elements of the event.

The webpage opens with a visually prominent Hero section that immediately presents the festival's identity through its name, dates, location, and a real-time countdown, framed over a full-screen video background. A fixed navigation bar allows users to move fluidly across the different areas of the page. The "About" section introduces the mood and atmosphere of the festival, providing some information about it, while the "Line-up & Performances" area organizes the program by day and stage, displaying each performer inside a structured card with their image and set time. It continues with the 'Stage' section that present an overview of the three festival stages, the 'Food, Drinks & Merchandise' section that gives a clear layout of available food, drinks and merchandise, and a catalogue of ticket options inserted into the 'Tickets' section. The page also includes a video performance preview, a location map to guide attendees, and a FAQ area addressing some common questions. A footer concludes the website with contact information and links to the festival's official social media channels.

2.1.2. Some screenshots of the web page

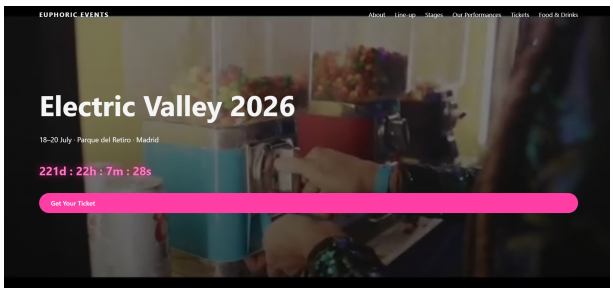


Figure 2.1: Hero section

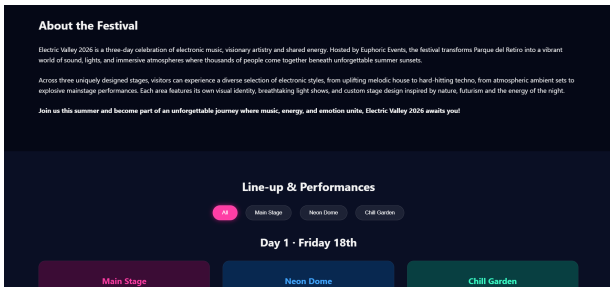


Figure 2.2: "About the festival" section

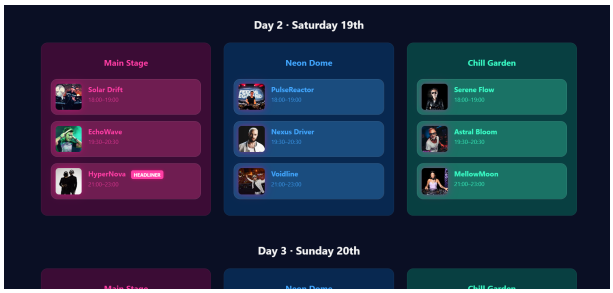


Figure 2.3: Line up & Performances section

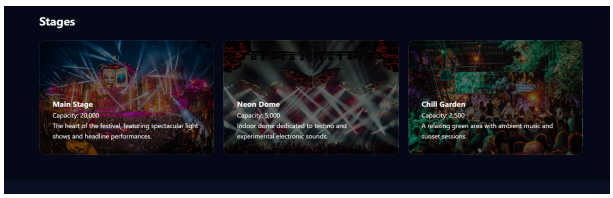


Figure 2.4: Stages section

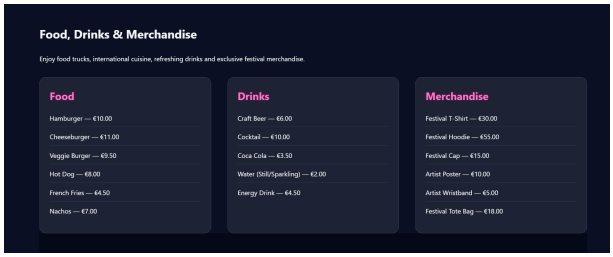


Figure 2.5: Food, Drinks and Merchandise section

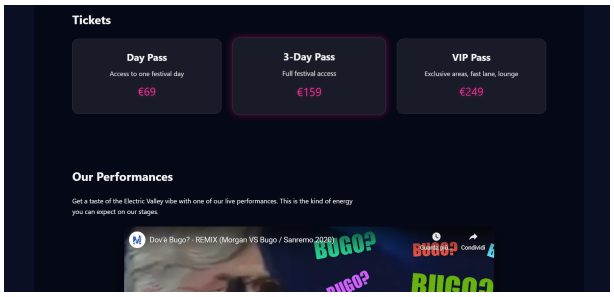


Figure 2.6: Tickets & Our Performances section

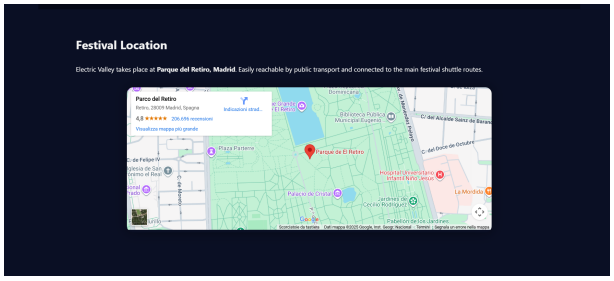


Figure 2.7: Festival Location section

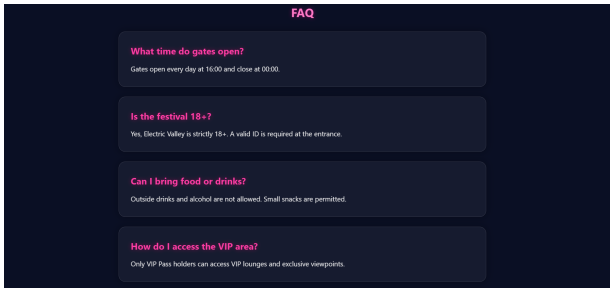


Figure 2.8: FAQ section

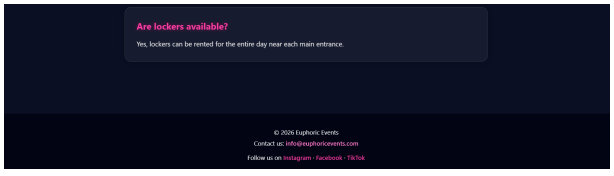


Figure 2.9: Footer section

2.1.3. Structure and Styling Discussion

The web page was built using HTML, CSS, and JavaScript to create a visually engaging but fully static page, as required by the assignment.

2.1.3.1. HTML

The HTML structure is fully modular and follows a component-based approach. Each major section is wrapped inside semantic containers such as `<header>`, `<section>` and `<footer>`, ensuring a clear logical separation of content. The Hero area is defined inside `<header class="hero">`, while the navigation bar is implemented through `<nav class="navbar">` and its internal `<ul class="nav-links">`. The Line-up is organised by grouping each day within `<div class="day-container">` and displaying stages using the grid container `<div class="stage-grid-visual">`. Individual performers are represented by `<div class="artist-card">`. The stage overview uses `<div class="stage-card">`, while food, drink and merchandise items are arranged in three structured columns built with `<div class="menu-column">`. Ticket options are implemented as `<div class="ticket-card">`, supported by an external tooltip element `<div id="ticket-tooltip">`. A video performance is embedded using `<div class="video-wrapper">`. Location and FAQ content rely on repeated structural blocks such as `<div class="faq-item">`. Overall, the HTML uses simple, reusable building blocks that makes the page clearer to read and easier to style.

2.1.3.2. CSS

The styling embraces a dark–neon aesthetic inspired by EDM festival visuals. A deep blue background, paired with bright pink and white accents, creates a futuristic and high-contrast look that feels energetic and immersive. The Hero section features a full-screen video background, implemented through `.hero-video`, together with a gradient overlay applied via `.hero::before` to keep the text clear and readable.

Across the page, the layout is structured using CSS Grid components such as `.stage-grid-visual`, `.menu-three-columns`, and `.ticket-grid`. Visual elements like `.artist-card`, `.menu-column`, and `.ticket-card` make use of glassmorphism effects, soft neon glows, and subtle blurring that reinforce the modern EDM feel. Stage-specific styles such as `.main-stage`, `.neon-dome`, and `.chill-garden` help distinguish the different environments of the festival, while classes like `.is-hidden` and `.single-stage` manage layout behaviour during filtering. Overall, the CSS keeps the visual identity consistent and helps create an atmosphere that feels cohesive and festival-like.

2.1.3.3. JavaScript

The JavaScript adds interactive behaviour to the page through a set of lightweight DOM manipulations. A countdown function regularly updates the `"#countdown"` element, while fade-in animations are triggered when elements marked with `".fade-in"` enter the user's viewport. A dynamic tooltip moves with the cursor when hovering over ticket cards, providing a more engaging interaction.

The Line-up section also includes a filtering mechanism: when a `.filter-btn` is selected, the script shows or hides `.stage-box` elements based on their stage classification (e.g. `"main-stage"`). It further refines the layout by hiding days without visible stages and centering days where only one stage is displayed. Together, these features support usability, keep the interface visually alive and uses a neon-inspired atmosphere to immerse the user in the world and aesthetic of EDM music.

2.2. Simple web page interacting with a server and a database

This project implements a simple web application that allows users to sign up and log in. The logic is divided into client-side validation, server-side processing, and database operations. The system ensures that only valid data is accepted and that proper feedback is returned to the user after each operation.

2.2.1. Client-Side Functionality

The web page contains two forms: one for signing up and one for logging in. The client-side JavaScript (script.js) handles all the interactions between the user and the server.

2.2.1.1. Input Validation

Before sending any data to the server, the script validates the user's input:

- All required fields must be filled.
- The email must contain "@".
- The password must be at least 4 characters long.

If any validation rule fails, the function stops immediately and displays a message such as:

- "Please fill all fields (client)"
- "Email is not valid (client)"
- "Password must be at least 4 characters (client)"

This prevents unnecessary server requests and improves usability.

2.2.1.2. Sending Requests to the Server

If the input passes validation, the script sends the data using the `fetch()` API:

- `/signup` for creating an account
- `/login` for accessing an existing account

Data is encoded using `URLSearchParams`, which keeps communication simple and compatible with Express.

2.2.1.3. Displaying Server Responses

The server always responds with a small JSON object:

- `{ "success": true or false, "message": "..." }`

The script reads this message and displays it below the corresponding form. This ensures that users receive real-time feedback after each operation.

2.2.2. Server-Side Functionality

The backend is implemented in Node.js using the Express framework (`server.js`). Its responsibilities include processing form submissions, validating them again, and interacting with the SQLite database.

2.2.2.1. Sign-Up Logic

When the user submits the sign-up form:

- The server receives the data via `POST /signup`.
- It checks that none of the fields are empty.
- It attempts to insert the new user into the database with the SQL query:
  - `INSERT INTO users (name, surname, email, password) VALUES (?, ?, ?, ?)`

- Because the `email` column is marked as `UNIQUE`, the database prevents duplicate registrations.

Possible Server Responses for Sign-Up:

- **Success:**
  "Welcome NAME SURNAME!"
- **Email already used:**
  "Sign-up failed (maybe email already used)"
- **Missing fields or other server issues:**
  "Fill all fields" or "Server error during sign-up"

These messages are passed back to the client and immediately displayed.

2.2.3. Login Logic

When a user attempts to log in:

- The client validates email and password.
- The server receives the data through `/login`.
- The server performs a database lookup:
  - `SELECT * FROM users WHERE email = ?  AND password = ?`
- If a matching user is found, login is successful.
- Otherwise, the server reports an authentication error.

Possible Server Responses for Login:

- **Success:**
  "Welcome back NAME SURNAME!"
- **Wrong credentials:**
  "Wrong email or password"
- **Missing fields:**
  "Fill email and password"

The client displays the returned message to inform the user of the result.

2.2.4. Some screenshots of the sign-up/log-in page

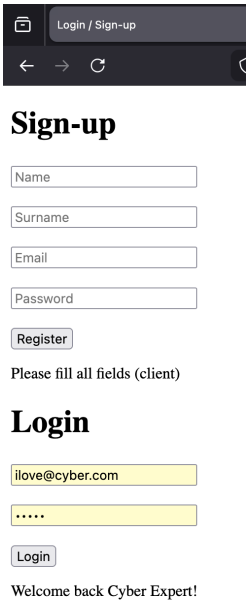


Figure 2.10: Login section when the login is confirmed

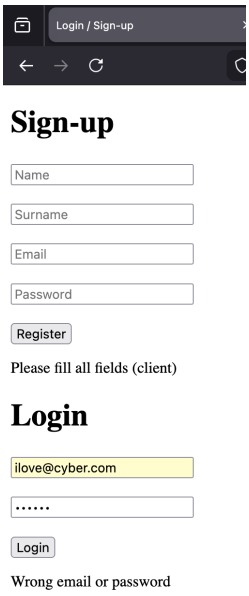


Figure 2.11: Login section when the login is invalid

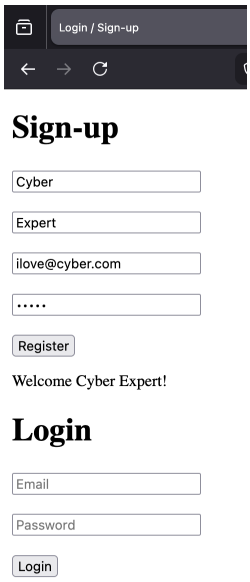


Figure 2.12: Sign-up section when the sign-up is done correctly

Figure 2.13: Sign-up section when the sign-up is done incorrectly