



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

EICTA Third Project Delivery – MongoDB & Web

Author(s): **Gabriella Bertolino**

Sofia Bulletti

Francesco Friolo

Domenico Pittari

Group Number: **05**

Academic Year: 2025-2026

Contents

Contents	i
1 MongoDB	1
1.1 Database structure	1
1.2 Queries	1
1.2.1 Insert new Festival (CREATE)	1
1.2.2 Apply discount to VIP tickets (UPDATE)	2
1.2.3 Add a new stand to an existing festival (UPDATE)	2
1.2.4 Remove expired tickets (DELETE)	3
1.2.5 Remove a festival (DELETE)	3
1.2.6 Budget-Friendly Weekend Tickets (PROJECTIONS AND LOGICAL OPERATORS)	3
1.2.7 Major European Venues (PROJECTIONS AND LOGICAL OPERATORS)	5
1.2.8 Premium VIP Experience (LIMIT, SORT AND COMPARISON OPERATORS)	6
1.2.9 Next 5 Upcoming Festivals (LIMIT, SORT AND COMPARISON OPERATORS)	7
1.2.10 Festivals with Sponsor Stands (ELEMENT QUERY OPERATORS)	8
1.2.11 Total Revenue per Festival (GROUP OPERATOR)	9
1.2.12 Ticket Types Breakdown (UNWIND AND GROUP BY)	10
1.2.13 Premium Customers (UNWIND, GROUP BY AND A FILTER) . .	11
1.2.14 Festivals with Both Large and Small Stages (TWO CONDITIONS EVALUATED SEPARATELY ON THE SUB-DOCUMENTS)	13
1.2.15 Medium-Capacity Lakeside Stages (TWO CONDITIONS EVALUATED SIMULTANEOUSLY ON THE SUB-DOCUMENTS)	14
2 Web	15

2.1	Promotional web page	15
2.1.1	Webpage content description	15
2.1.2	Structure and Styling	19
2.2	Web page interacting with a server and a database	20
2.2.1	Web page contents	20
2.2.2	Client-Side logic	21
2.2.3	Server Configuration	22
2.2.4	Database Setup	22
2.2.5	Sign-Up Endpoint	22
2.2.6	Login Endpoint	23
2.2.7	Styling	23

1 | MongoDB

1.1. Database structure

From the ER model we chose the following entities, along with their attributes:

- Festival: name, start_date, end_date, location
- Stage: ID, name, capacity, location_description
- Stand: name, sponsor
- Ticket: QR_code, validFrom, validTo, price, type
- Person: email, phone_number, full_name

In the MongoDB database we created a unique collection called *festivals*, which stores one document per festival. Each festival document has the implicit *_id: ObjectId* created by MongoDB, a *festival* object with its attributes, and three embedded arrays—*stages*, *stands*, and *tickets*. Each ticket embeds a single *validity* object defining a date range and a *person* array.

1.2. Queries

1.2.1. Insert new Festival (CREATE)

This query creates a new festival event with stages, stands, and initial tickets.

```

1 db.festival.insertOne({
2   _id: "FEST012",
3   festivalName: "Alpine Beats 2026",
4   startDate: ISODate("2026-01-15"),
5   endDate: ISODate("2026-01-17"),
6   location: "Zurich, CH",
7   stages: [

```

```

8     { _id: "STAGE023", stageName: "Mountain Peak", capacity: 3500,
9      ↵   locationDescription: "Alpine lodge" }],  

10    stands: [  

11      { _id: "STAND023", standName: "Swiss Treats", sponsor: "Alps Food Co" }],  

12      tickets: [  

13        { _id: "TKT045", qrCode: "QR-AB-0001", price: 85, ticketType: "DAY",
14         ↵   validFrom: ISODate("2026-01-15"), validTo: ISODate("2026-01-15"),
15         ↵   person: { _id: "PERS023", email: "hans.m@example.ch", phoneNumber:
16         ↵   "+41-44-555-1111", fullName: "Hans Mueller" }]}]);

```

Output:

```

1 { "acknowledged": true,  

2   "insertedId": "FEST012"}

```

1.2.2. Apply discount to VIP tickets (UPDATE)

Reduces the price of all VIP tickets by 10% for early bird promotion.

```

1 db.festival.updateMany(  

2   { "tickets.ticketType": "VIP" },  

3   { $mul: { "tickets.$[ticket].price": 0.9 } },  

4   { arrayFilters: [{ "ticket.ticketType": "VIP" }]});

```

Output:

```

1 { "acknowledged": true,  

2   "insertedId": null,  

3   "matchedCount": 11,  

4   "modifiedCount": 11,  

5   "upsertedCount": 0 }

```

1.2.3. Add a new stand to an existing festival (UPDATE)

This query adds a new merchandise stand to the Sunwave Fest 2025. The output indicates that one festival was found and the targeted stand was added.

```

1 db.festivals.updateOne(  

2   { _id: "FEST001" },  

3   { $push: { stands: { _id: "STAND024", standName: "Sunwave Souvenirs",
4     ↵   sponsor: "Festival Memories Inc" }}}} );

```

Output:

```

1 { "acknowledged": true,
2   "insertedId": null,
3   "matchedCount": 1,
4   "modifiedCount": 1,
5   "upsertedCount": 0 }
```

1.2.4. Remove expired tickets (DELETE)

This query removes all tickets for festivals that have already ended. The output indicates that four tickets were successfully removed.

```

1 db.festival.deleteMany(
2   { endDate: { $lt: ISODate("2025-07-01") }});
```

Output:

```

1 { "acknowledged": true,
2   "deletedCount": 4 }
```

1.2.5. Remove a festival (DELETE)

This query completely removes a festival that has been cancelled. The output indicates that the festival was found and it was successfully removed.

```
1 db.festival.deleteOne({ _id: "FEST012" });
```

Output:

```

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

1.2.6. Budget-Friendly Weekend Tickets (PROJECTIONS AND LOGICAL OPERATORS)

This query retrieves festival names and weekend ticket details where price is between 100-140 OR location is in the US, showing only relevant fields. The query also limits the output to 2 answers.

```

1 db.festival.find(
2   { $or: [
3     { "tickets.ticketType": "WEEKEND", "tickets.price": { $gte: 100, $lte:
4       → 140 } },
5     { location: /US$/ } ],
6   projection: { name: 1, "tickets.type": 1, "tickets.price": 1 } })
```

```

5      "tickets.ticketType": "WEEKEND" },
6      { festivalName: 1,
7       location: 1,
8       "tickets.$": 1 }
9   ).limit(2);

```

Output:

```

1  [{ "_id": "FEST001",
2   "festivalName": "Sunwave Fest 2025",
3   "location": "Lisbon, PT",
4   "tickets": [
5     { "_id": "TKT003",
6       "qrCode": "QR-SUN-0003",
7       "price": 120,
8       "ticketType": "WEEKEND",
9       "validFrom": { "$date": "2025-07-05T00:00:00Z" },
10      "validTo": { "$date": "2025-07-07T00:00:00Z" },
11      "person": {
12        "_id": "PERS002",
13        "email": "marco.v@example.com",
14        "phoneNumber": "+351-922222222",
15        "fullName": "Marco Vieira" }}}},
16  { "_id": "FEST002",
17   "festivalName": "Aurora Sounds 2025",
18   "location": "Helsinki, FI",
19   "tickets": [
20     { "_id": "TKT007",
21       "qrCode": "QR-AUR-0003",
22       "price": 135,
23       "ticketType": "WEEKEND",
24       "validFrom": { "$date": "2025-08-15T00:00:00Z" },
25       "validTo": { "$date": "2025-08-17T00:00:00Z" },
26       "person": {
27         "_id": "PERS004",
28         "email": "mikko.l@example.fi",
29         "phoneNumber": "+358-402222222",
30         "fullName": "Mikko Laine"
31       }}}},
32

```

1.2.7. Major European Venues (PROJECTIONS AND LOGICAL OPERATORS)

This query finds festivals in Europe with stages having capacity over 4000, projecting only essential information.

```

1 db.festival.find(
2   { $and: [
3     { location: { $in: ["Lisbon, PT", "Helsinki, FI", "Milan, IT"] } },
4     { "stages.capacity": { $gt: 4000 }}}},
5   { festivalName: 1,
6    location: 1,
7    startDate: 1,
8    "stages.stageName": 1,
9    "stages.capacity": 1 });

```

Output:

```

1 [ { "_id": "FEST001",
2   "festivalName": "Sunwave Fest 2025",
3   "startDate": { "$date": "2025-07-05T00:00:00Z" },
4   "location": "Lisbon, PT",
5   "stages": [
6     { "stageName": "Main Stage",
7      "capacity": 5000 },
8     { "stageName": "Indie Stage",
9      "capacity": 2500 }]},
10 { "_id": "FEST002",
11   "festivalName": "Aurora Sounds 2025",
12   "startDate": { "$date": "2025-08-15T00:00:00Z" },
13   "location": "Helsinki, FI",
14   "stages": [
15     { "stageName": "Aurora Dome",
16       "capacity": 4200 },
17     { "stageName": "Lakeside",
18       "capacity": 2600 }]},
19 { "_id": "FEST011",
20   "festivalName": "Valley Groove 2025",
21   "startDate": { "$date": "2025-09-12T00:00:00Z" },
22   "location": "Milan, IT",
23   "stages": [

```

```

24     { "stageName": "Duomo Stage",
25         "capacity": 5000 },
26     { "stageName": "Navigli Stage",
27         "capacity": 2400 }}]
```

1.2.8. Premium VIP Experience (LIMIT, SORT AND COMPARISON OPERATORS)

This query returns the top 3 most expensive VIP tickets, sorted by price descending.

```

1 db.festival.aggregate([
2     { $unwind: "$tickets" },
3     { $match: { "tickets.ticketType": "VIP", "tickets.price": { $gte: 200 } } },
4     { $sort: { "tickets.price": -1 } },
5     { $limit: 3 },
6     { $project: {
7         festivalName: 1,
8         "tickets.qrCode": 1,
9         "tickets.price": 1,
10        "tickets.person.fullName": 1 }}}]);
```

Output:

```

1 [ { "_id": "FEST006",
2     "festivalName": "City Lights Live 2025",
3     "tickets": {
4         "qrCode": "QR-CL-0002",
5         "price": 240,
6         "person": { "fullName": "Emma Davis" }}}},
7     { "_id": "FEST004",
8     "festivalName": "Coast Vibes 2025",
9     "tickets": {
10        "qrCode": "QR-CV-0004",
11        "price": 230,
12        "person": { "fullName": "Noah Baker" }}}},
13     { "_id": "FEST009",
14     "festivalName": "Harbor Harmonies 2025",
15     "tickets": {
16        "qrCode": "QR-HH-0002",
17        "price": 225,
18        "person": { "fullName": "Harper Clark" }}}]
```

1.2.9. Next 5 Upcoming Festivals (LIMIT, SORT AND COMPARISON OPERATORS)

This query lists the next 5 festivals scheduled after a specific date, sorted by start date.

```

1 db.festival.find(
2   { startDate: { $gte: ISODate("2025-07-03") } } )
3   .sort({ startDate: 1 })
4   .limit(5)
5   .projection({ festivalName: 1, startDate: 1, endDate: 1, location:1 });

```

Output:

```

1 [ { "_id": "FEST001",
2   "festivalName": "Sunwave Fest 2025",
3   "startDate": { "$date": "2025-07-05T00:00:00Z" },
4   "endDate": { "$date": "2025-07-07T00:00:00Z" },
5   "location": "Lisbon, PT" },
6 { "_id": "FEST009",
7   "festivalName": "Harbor Harmonies 2025",
8   "startDate": { "$date": "2025-07-18T00:00:00Z" },
9   "endDate": { "$date": "2025-07-20T00:00:00Z" },
10  "location": "Seattle, US" },
11 { "_id": "FEST007",
12  "festivalName": "Mountain Echo 2025",
13  "startDate": { "$date": "2025-08-01T00:00:00Z" },
14  "endDate": { "$date": "2025-08-03T00:00:00Z" },
15  "location": "Denver, US" },
16 { "_id": "FEST002",
17  "festivalName": "Aurora Sounds 2025",
18  "startDate": { "$date": "2025-08-15T00:00:00Z" },
19  "endDate": { "$date": "2025-08-17T00:00:00Z" },
20  "location": "Helsinki, FI" },
21 { "_id": "FEST004",
22  "festivalName": "Coast Vibes 2025",
23  "startDate": { "$date": "2025-09-01T00:00:00Z" },
24  "endDate": { "$date": "2025-09-03T00:00:00Z" },
25  "location": "San Diego, US" }]

```

1.2.10. Festivals with Sponsor Stands (ELEMENT QUERY OPERATORS)

This query finds all festivals that have at least one stand with a sponsor field defined.

```

1 db.festival.find(
2   { "stands.sponsor": { $exists: true, $ne: null } },
3   { festivalName: 1,
4     location: 1,
5     stands: 1 }
6 ).limit(3);

```

Output:

```

1 [ { "_id": "FEST001",
2   "festivalName": "Sunwave Fest 2025",
3   "location": "Lisbon, PT",
4   "stands": [
5     { "_id": "STAND001",
6       "standName": "Tasty Bites",
7       "sponsor": "Gusto Inc." },
8     { "_id": "STAND002",
9       "standName": "Band Merch",
10      "sponsor": "MerchCo" }]},
11 { "_id": "FEST002",
12   "festivalName": "Aurora Sounds 2025",
13   "location": "Helsinki, FI",
14   "stands": [
15     { "_id": "STAND003",
16       "standName": "Finn Flavors",
17       "sponsor": "Nordic Foods" },
18     { "_id": "STAND004",
19       "standName": "Glow Merch",
20       "sponsor": "Polar Merch LLC" }]},
21 { "_id": "FEST003",
22   "festivalName": "Desert Beats 2025",
23   "location": "Phoenix, US",
24   "stands": [
25     { "_id": "STAND005",
26       "standName": "Cool Drinks",
27       "sponsor": "H2O Co" },

```

```

28     { "_id": "STAND006",
29         "standName": "Desert Merch",
30         "sponsor": "Sunshine Merch" }]}]
```

1.2.11. Total Revenue per Festival (GROUP OPERATOR)

This aggregation pipeline calculates the total ticket revenue for each festival.

```

1 db.festival.aggregate([
2     { $unwind: "$tickets" },
3     { $group: {
4         _id: "$festivalName",
5         totalRevenue: { $sum: "$tickets.price" },
6         ticketsSold: { $count: {} },
7         location: { $first: "$location" } } },
8     { $sort: { totalRevenue: -1 } } ]);
```

Output:

```

1 [ { "_id": "City Lights Live 2025",
2     "totalRevenue": 495,
3     "ticketsSold": 4,
4     "location": "New York, US" },
5     { "_id": "Coast Vibes 2025",
6         "totalRevenue": 455,
7         "ticketsSold": 4,
8         "location": "San Diego, US" },
9     { "_id": "Harbor Harmonies 2025",
10        "totalRevenue": 443,
11        "ticketsSold": 4,
12        "location": "Seattle, US" },
13     { "_id": "Aurora Sounds 2025",
14         "totalRevenue": 430,
15         "ticketsSold": 4,
16         "location": "Helsinki, FI" },
17     { "_id": "Desert Beats 2025",
18         "totalRevenue": 415,
19         "ticketsSold": 4,
20         "location": "Phoenix, US" },
21     { "_id": "Prairie Pulse 2025",
22         "totalRevenue": 405,
```

```

23     "ticketsSold": 4,
24     "location": "Chicago, US" },
25   { "_id": "Mountain Echo 2025",
26     "totalRevenue": 402,
27     "ticketsSold": 4,
28     "location": "Denver, US" },
29   { "_id": "Valley Groove 2025",
30     "totalRevenue": 397,
31     "ticketsSold": 4,
32     "location": "Milan, IT" },
33   { "_id": "Sunwave Fest 2025",
34     "totalRevenue": 395,
35     "ticketsSold": 4,
36     "location": "Lisbon, PT" },
37   { "_id": "Forest Jam 2025",
38     "totalRevenue": 378,
39     "ticketsSold": 4,
40     "location": "Portland, US" },
41   { "_id": "RiverRhythm 2025",
42     "totalRevenue": 373,
43     "ticketsSold": 4,
44     "location": "Nashville, US" }]

```

1.2.12. Ticket Types Breakdown (UNWIND AND GROUP BY)

This query shows how many tickets of each type have been sold across all festivals.

```

1 db.festival.aggregate([
2   { $unwind: "$tickets" },
3   { $group: {
4     _id: "$tickets.ticketType",
5     count: { $count: {} },
6     averagePrice: { $avg: "$tickets.price" },
7     totalRevenue: { $sum: "$tickets.price" } } },
8   { $sort: { totalRevenue: -1 } } ]);

```

Output:

```

1 [ { "_id": "VIP",
2   "count": 11,
3   "averagePrice": 211.8181818181818,

```

```

4     "totalRevenue": 2330 },
5   { "_id": "WEEKEND",
6     "count": 11,
7     "averagePrice": 131.8181818181818,
8     "totalRevenue": 1450 },
9   { "_id": "DAY",
10    "count": 11,
11    "averagePrice": 54.36363636363637,
12    "totalRevenue": 598 },
13   { "_id": "MERCH_VOUCHER",
14    "count": 11,
15    "averagePrice": 19.09090909090909,
16    "totalRevenue": 210 }]

```

1.2.13. Premium Customers (UNWIND, GROUP BY AND A FILTER)

This query identifies customers who have spent more than 200*totalontickets*.

```

1 db.festival.aggregate([
2   { $unwind: "$tickets" },
3   { $group: {
4     _id: "$tickets.person._id",
5     personName: { $first: "$tickets.person.fullName" },
6     email: { $first: "$tickets.person.email" },
7     totalSpent: { $sum: "$tickets.price" },
8     ticketCount: { $count: {} } },
9   { $match: { totalSpent: { $gt: 200 } } },
10  { $sort: { totalSpent: -1 } } ]);

```

output:

```

1 [ { "_id": "PERS014",
2   "personName": "Lucas Garcia",
3   "email": "lucas.g@example.com",
4   "totalSpent": 335,
5   "ticketCount": 2 },
6   { "_id": "PERS022",
7   "personName": "Luca Bianchi",
8   "email": "luca.b@example.it",
9   "totalSpent": 330,

```

```
10      "ticketCount": 2 },
11  { "_id": "PERS002",
12    "personName": "Marco Vieira",
13    "email": "marco.v@example.com",
14    "totalSpent": 320,
15    "ticketCount": 2 },
16  { "_id": "PERS016",
17    "personName": "Elijah Moore",
18    "email": "elijah.m@example.com",
19    "totalSpent": 315,
20    "ticketCount": 2 },
21  { "_id": "PERS011",
22    "personName": "Emma Davis",
23    "email": "emma.d@example.com",
24    "totalSpent": 310,
25    "ticketCount": 2 },
26  { "_id": "PERS008",
27    "personName": "Noah Baker",
28    "email": "noah.b@example.com",
29    "totalSpent": 285,
30    "ticketCount": 2 },
31  { "_id": "PERS017",
32    "personName": "Harper Clark",
33    "email": "harper.c@example.com",
34    "totalSpent": 283,
35    "ticketCount": 2 },
36  { "_id": "PERS003",
37    "personName": "Liisa Korhonen",
38    "email": "liisa.k@example.fi",
39    "totalSpent": 270,
40    "ticketCount": 2 },
41  { "_id": "PERS019",
42    "personName": "Amelia Scott",
43    "email": "amelia.s@example.com",
44    "totalSpent": 260,
45    "ticketCount": 2 },
46  { "_id": "PERS006",
47    "personName": "Ethan Wood",
48    "email": "ethan.w@example.com",
49    "totalSpent": 235,
```

```

50     "ticketCount": 2 },
51   { "_id": "PERS010",
52     "personName": "William Harris",
53     "email": "will.h@example.com",
54     "totalSpent": 210,
55     "ticketCount": 2 }]

```

1.2.14. Festivals with Both Large and Small Stages (TWO CONDITIONS EVALUATED SEPARATELY ON THE SUB-DOCUMENTS)

This query finds festivals that have at least one stage with capacity > 5000 AND at least one stage with capacity < 3000.

```

1 db.festival.find({
2   $and: [
3     { "stages.capacity": { $gt: 5000 } },
4     { "stages.capacity": { $lt: 3000 } }]},
5   { festivalName: 1,
6    location: 1,
7    stages: 1 });

```

Output:

```

1 [ { "_id": "FEST008",
2   "festivalName": "RiverRhythm 2025",
3   "location": "Nashville, US",
4   "stages": [
5     { "_id": "STAGE015",
6       "stageName": "Bridge Stage",
7       "capacity": 5200,
8       "locationDescription": "Riverside" },
9     { "_id": "STAGE016",
10      "stageName": "Honkytonk Stage",
11      "capacity": 2800,
12      "locationDescription": "Downtown" }]},
13   { "_id": "FEST010",
14     "festivalName": "Prairie Pulse 2025",
15     "location": "Chicago, US",
16     "stages": [
17       { "_id": "STAGE019" },
18       { "_id": "STAGE020" }]}
19 ]

```

```

18     "stageName": "Wind Stage",
19     "capacity": 5200,
20     "locationDescription": "Park lawn" },
21   { "_id": "STAGE020",
22     "stageName": "City Stage",
23     "capacity": 2600,
24     "locationDescription": "Downtown plaza" }
25   }]

```

1.2.15. Medium-Capacity Lakeside Stages (TWO CONDITIONS EVALUATED SIMULTANEOUSLY ON THE SUB-DOCUMENTS)

This query finds festivals with stages that simultaneously have capacity between 2500-3000 AND contain "Lake" or "lake" in the location description.

```

1 db.festival.find({
2   stages: {
3     $elemMatch: {
4       capacity: { $gte: 2500, $lte: 3000 },
5       locationDescription: { $regex: /lake/i } }
6   }, {
7     festivalName: 1,
8     location: 1,
9     "stages.$": 1 });

```

Output:

```

1 [ { "_id": "FEST002",
2   "festivalName": "Aurora Sounds 2025",
3   "location": "Helsinki, FI",
4   "stages": [ {
5     "_id": "STAGE004",
6     "stageName": "Lakeside",
7     "capacity": 2600,
8     "locationDescription": "By the lake" }
9   ] ]

```

2 | Web

2.1. Promotional web page

As part of this project, we designed a static promotional web page for Euphoric Events, focusing on a specific festival instance called SunWave Festival 2026.

In this section of the report, we describe the contents of the web page, and we discuss the structure and styling choices. We also include screenshots of the final design to illustrate how the different components work together to promote SunWave Festival 2026.

2.1.1. Webpage content description

The webpage is structured into five major sections plus a footer. Below is a description of each part.

Navigation bar: a fixed, semi-transparent navigation bar appears at the top of the page. It includes links that take you to each section.

Hero section: acts as the visual entry point of the festival page. It includes:

- A background with animated waves
- A large title and subtitle
- The festival's date and location
- A call-to-action button ("Get Your Tickets")
- A floating particle effect generated dynamically with JavaScript
- Animations defined in CSS, and particles created in JavaScript.

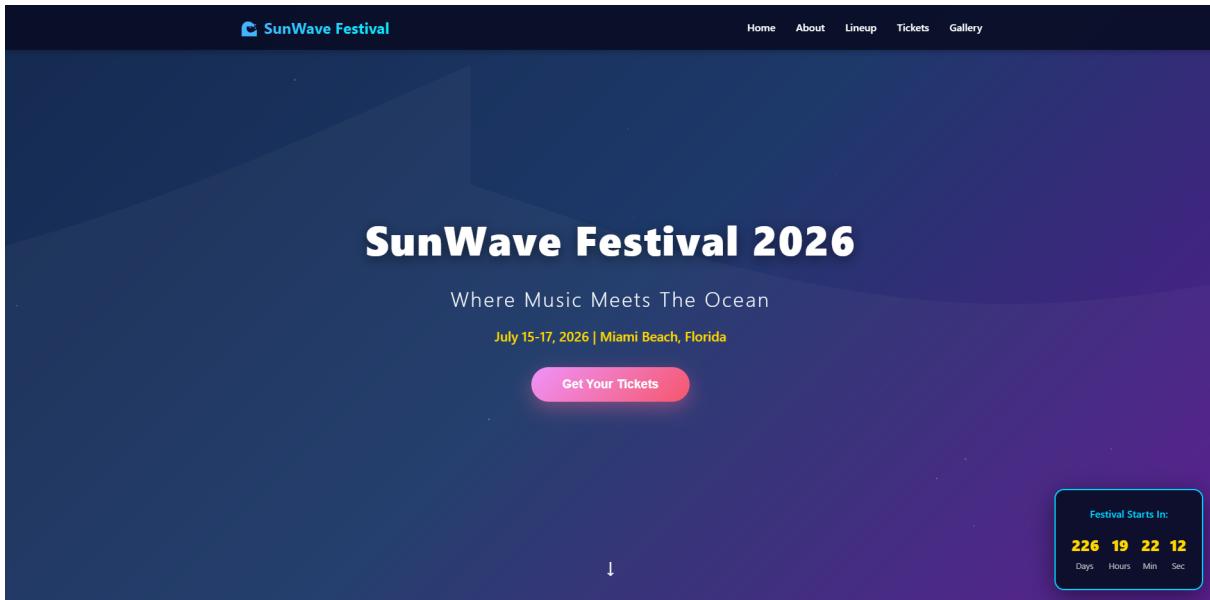


Figure 2.1: Navigation bar and Hero section

About section: provides an overview of SunWave Festival. It contains:

- A text introduction
- Three “blocks”: 100+ Artists, Art Installations and Gourmet Food
- Each feature includes an emoji, title, and description

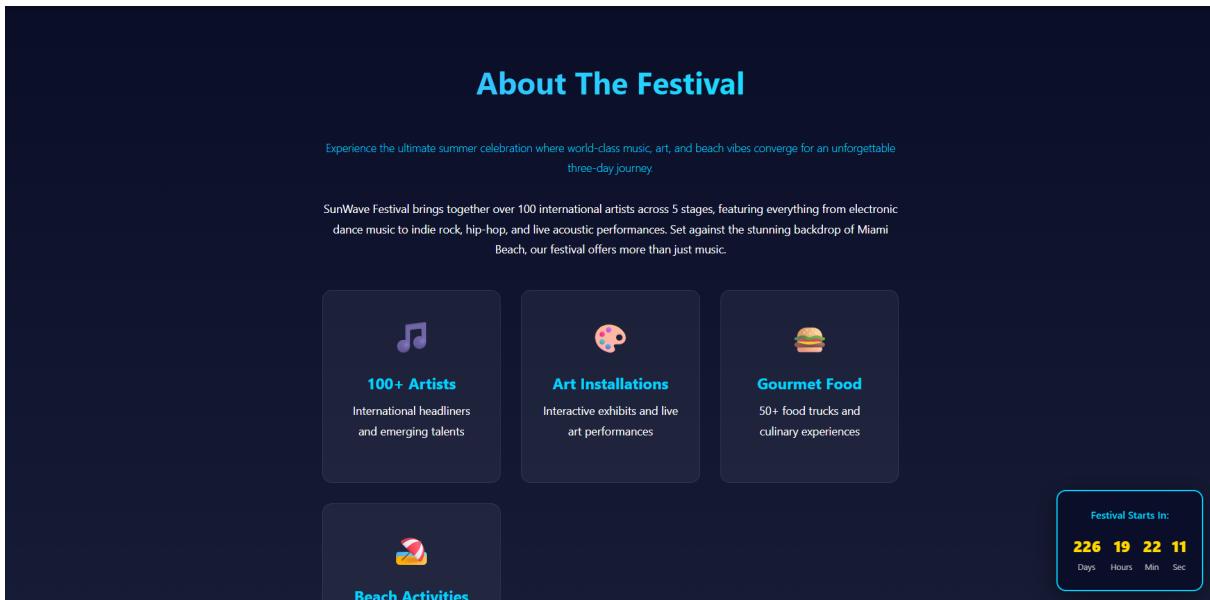


Figure 2.2: About section

Lineup section: shows the festival artists for each of the three days. The user can switch between days using JavaScript-driven buttons, which show/hide the corresponding lineup container via the showDay() function. Artist cards include name, genre, performance time, and stage.

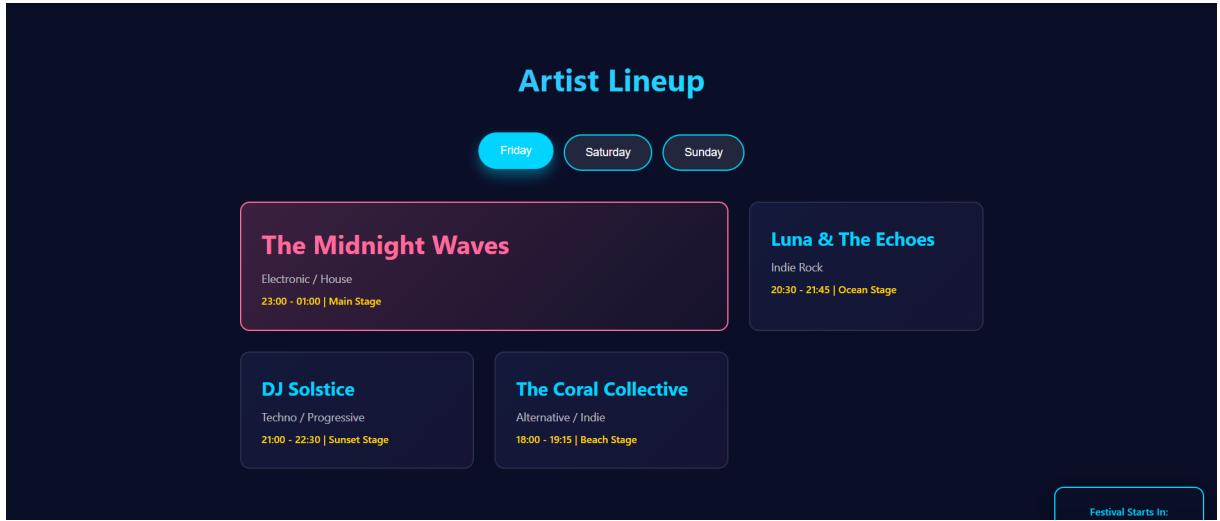


Figure 2.3: Lineup section

Tickets section: three types of tickets are presented: General Admission, VIP Experience, and Ultimate VIP. Each ticket contains:

- Its name
- Price
- A list of features
- A “Buy Now” button.

Clicking on the “Buy Now” button triggers a JavaScript alert simulating a purchase.

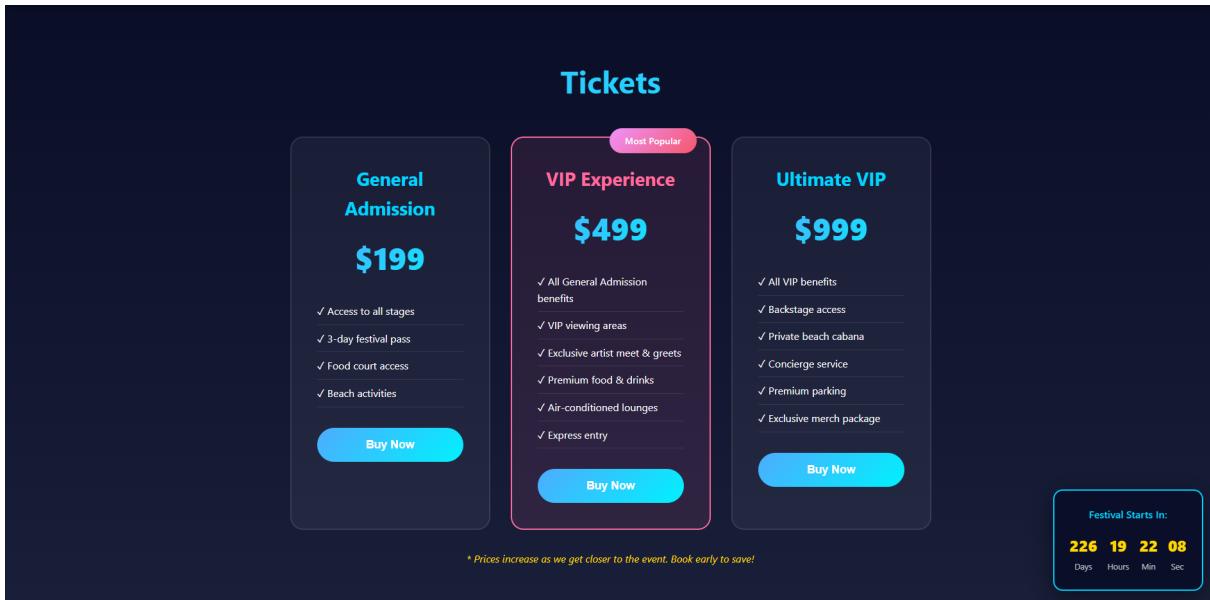


Figure 2.4: Ticket section

Gallery section: displays six images sourced from Unsplash and embedded via URLs. Hovering over an image reveals a semi-transparent overlay with a caption. Clicking triggers a JavaScript message simulating a lightbox feature. This section also includes an embedded aftermovie video.

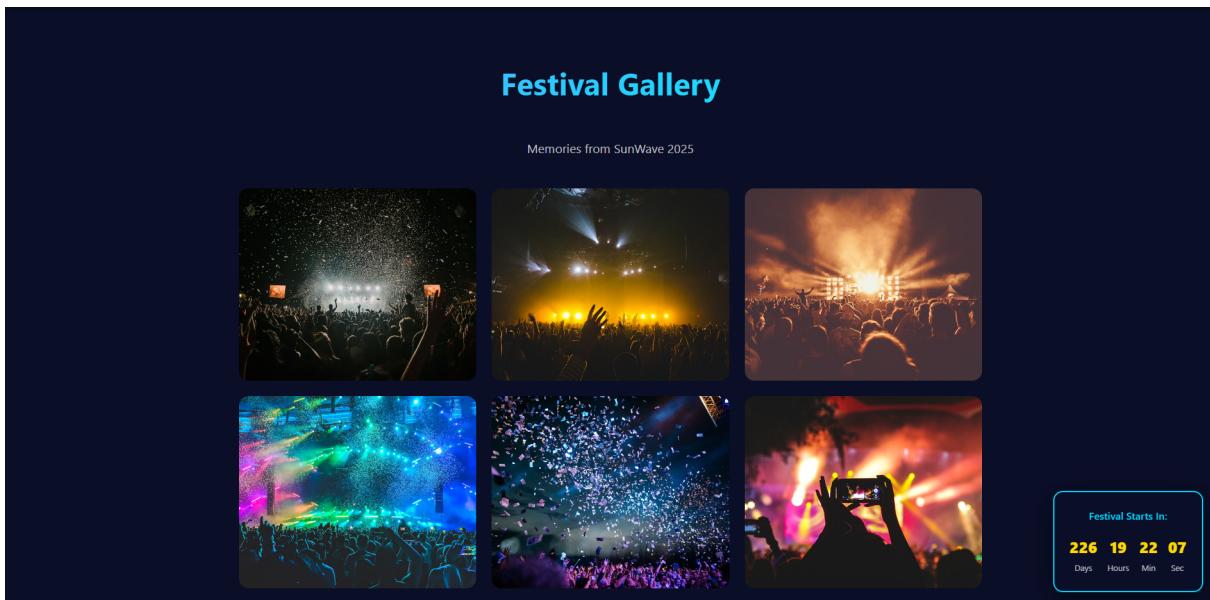


Figure 2.5: Gallery section

Footer and Countdown widget: the footer contains: festival information, quick links, contact details, and subscription to the newsletter (handled client-side with val-

idation). Additionally, a countdown widget shows the time remaining until July 15, 2026. JavaScript updates the countdown every second using the `updateCountdown()` function.

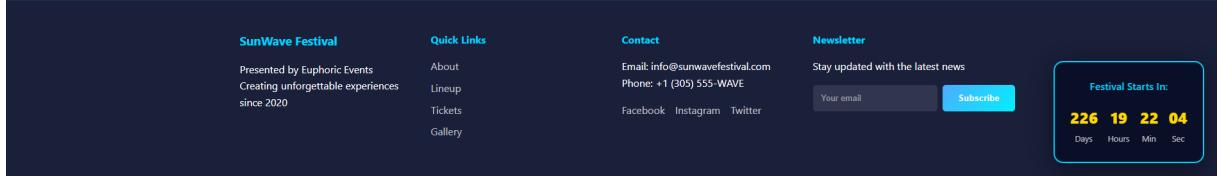


Figure 2.6: Footer

2.1.2. Structure and Styling

This section provides an overview of the organizational choices, visual design strategies, and interactive elements that define the look and behavior of the web page.

HTML:

- <nav> for the navigation bar
- <section> for content blocks (hero, about, lineup, tickets, gallery)
- <footer> for page closure.

CSS:

- Color palette and variables
- Animations (fade-in transitions, hover transformations, movement of the wave-like background, floating particles, navbar hide/reappear effect)
- Layout techniques.

JavaScript:

- Smooth section scroll
- Lineup day switching
- Countdown timer
- Element animation on scroll
- Interactive gallery
- Floating particles

2.2. Web page interacting with a server and a database

We also implemented a web application that allows users to sign up and log in using a client-side interface that communicates with a Node.js + Express server. The back-end validates the input data, queries an SQLite database, and returns structured responses. The client-side handles form submission, displays success/error messages, and enforces basic validations before sending requests.

2.2.1. Web page contents

The application consists of a single main page that contains two forms: a login form and a sign-up form, shown alternatively using JavaScript. The HTML structure is defined in index.html and contains:

- A login box with fields for email and password
- A sign-up box with fields for first name, last name, email, password, and confirm password
- A message container used to display server responses
- Links allowing to switch between login and sign-up modes.

Both forms rely on JavaScript event listeners for handling submission logic and interaction with the server.

2.2.2. Client-Side logic

All client logic is implemented in script.js and includes:

Form switching: the functions showLogin() and showSignup() switch the visibility of the two form boxes by adding or removing the .hidden CSS class.

Form submission and validation: two main listeners handle the submissions: loginForm and signupForm.

Login Form: prevents default submission, performs **client-side validation** (email format via regex $/^[\^s@]+@[^\^s@]+\.\[^s@]+$/$ and password length ≥ 6 characters), collects email and password, sends a POST request to /api/login, and displays a message upon a valid response or an error.

Signup Form: prevents default submission, performs **client-side validation** including:

- Name and surname validation (minimum 2 characters, only letters and spaces via regex $/^[\^zA-Z\s]+$/$)
- Email format validation (same regex as login)
- Password length validation (minimum 6 characters)
- Password confirmation match verification

After validation, it sends a POST request to /api/signup, shows a success message, and returns to the login form after 2 seconds.

Defense in Depth approach: both client-side and server-side validations are implemented. Client-side validation provides immediate user feedback and reduces unnecessary server requests, while server-side validation ensures security even if client-side checks are bypassed (e.g., via direct API calls). The backend uses the same regular expressions to verify email format and validates all input fields before processing authentication or registration requests.

User Feedback: the functions showMessage() and hideMessage() update the visual message component. Success messages appear with green styling, while validation errors and authentication failures appear with red styling, providing clear user feedback for all operations.

2.2.3. Server Configuration

The back-end of the application is implemented in `server.js` using Node.js and Express. It listens on port 3000 and automatically establishes a connection to the SQLite database, preparing the environment before handling any requests.

2.2.4. Database Setup

When the server starts, it creates the user table if it does not already exist. The table includes fields for basic personal data (name, surname, email), a password, and an automatic timestamp. If the table is empty, the server also inserts a set of default users, such as John Doe (`john@example.com`), Jane Smith (`jane@example.com`), and Mike Johnson (`mike@example.com`), to allow immediate testing of the authentication flow.

2.2.5. Sign-Up Endpoint

The sign-up route handles the creation of new accounts by validating the user's input. It checks that all fields are filled in, that names contain only letters, the email is correctly formatted and not already in use, and that the password meets minimum length requirements (at least 6 characters). If everything is valid, the user is added to the database and a confirmation message is returned.

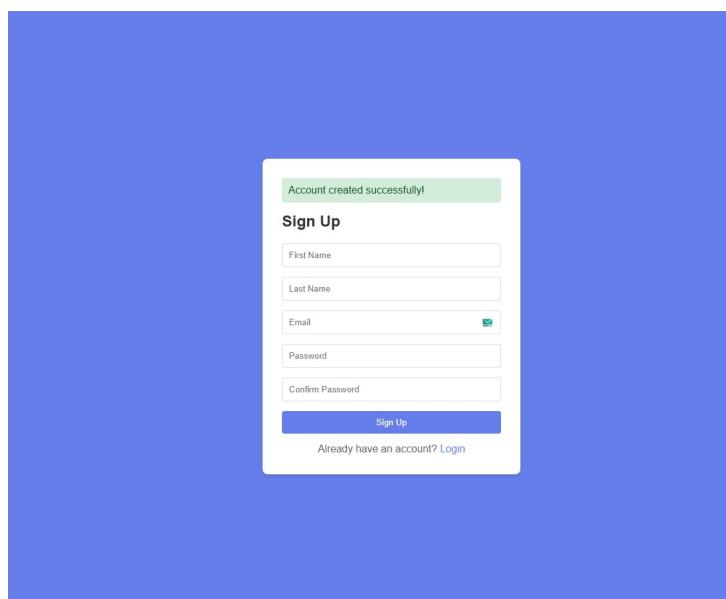


Figure 2.7: Sign-up success message

2.2.6. Login Endpoint

The server ensures that both email and password are provided, verifies the email format, and then searches for a matching user in the database. If the stored password matches the one provided, the server responds with a success message and the user's information. Invalid credentials trigger an error message.

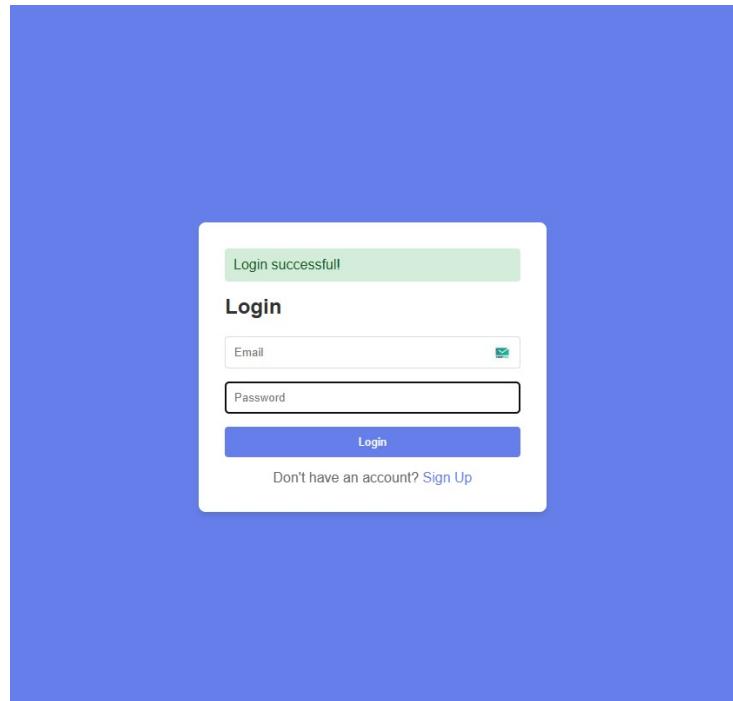


Figure 2.8: Login success message

2.2.7. Styling

Styling is defined in style.css. Key aspects include:

- A centered white container with rounded corners and a shadow
- Color-coded messages