Documentation

1. How to Build and Run this Application

   Requirement:

   - node.js v22+
   - npm

   Running in Local:

   ```
   1.
   cd /express
   npm install
   npm start

   2.
   cd /react
   npm install
   npm start
   ```

2. Live Link

   https://vancouver-public-art.onrender.com/

3. Dataset and Project Description

   This term project uses **Vancouver Public Art** from Vancouver Open Data Portal.

   I named it **Van Artwork explorer.**

   - Expected Users:
     - Tourists and residents who are interested in their neighbourhood.
     - Art students or local artists who want to study installed arts.
   - Problem this application solves
     - Helping people easily browse public arts in Vancouver without searching individually.

4. How to use application

   a. Artwork List
      - Displays artworks in a paginated table(10 per page)
      - Shows title, Location, and status
      - Clicking a row opens the detail view below.

   b. Filtering
      - In place
      - No longer in place
      - Deaccessioned

   c. Detail View
      - Displays:

        - Title
        - Status
        - Year of installation
        - Artist Statement
        - Location
        - Photo
    d. Create New Art
        - Click ‘Add Art’
        - Fill out the form
    e. Edit Art
        - Click ‘Edit’
        - Fill out the form
5. Attributions
    - https://www.sqlitetutorial.net/sqlite-functions/sqlite-lower/
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/keys
    - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/values
    - https://www.google.com/search?q=reason+why+react+user+prev&newwindow=1&sca_esv=75e8315709cd5118&sxsrf=AE3TifMJSPU7cgRZxxgg0C95VAEzy02WW6BgYIARABGAGSBwQzNC41oAfs3wGyBwQyNi41uAfsE8IHCDAuOC4zMC4xyAe1AQ&sclient=gws-wiz-serp
    - https://expressjs.com/en/guide/migrating-5.html
6. AI Usage

AI Usage_Bitna Lee

Tool Used: ChatGPT (GPT-5.1)

Purpose:

To clarify several Express.js and backend-architecture behaviors.

My goal was to understand proper error-handling patterns, Express 5 wildcard routing behavior, and best practices for transaction-based error control in SQLite.

Prompt History

I asked the AI the following questions (originally asked in Korean, translated into English):

1. “Is it a best practice to wrap every backend controller function in a try/catch block?”

2. “In Express 5, wildcard routes like * do not work anymore. I saw /*path syntax in the docs—should my code use that, and is my implementation correct?”

3. “When using SQL transactions, what is the best way to implement backend error handling so that rollback always works safely?”

AI Responses

1. Whether all controllers must use try/catch

The AI explained that:

- It is **not required** to wrap *all* controllers in try/catch.
- For **database-dependent operations** (create/update/delete), try/catch **is recommended** to catch DB errors.
- For **simple GET controllers**, a try/catch block is optional because these operations rarely throw runtime exceptions unless the model throws manually.
- A better pattern is:
  - keep **try/catch in mutating endpoints** (POST, PATCH, PUT, DELETE)
  - keep **middleware validations** handling wrong inputs before controller runs
  - use a **global error handler** for unexpected failures

This clarified that my original structure was acceptable and followed real-world practices.

---

2. Express 5 wildcard route not working (*, /*path, etc.)

The AI explained that:

- Express 5 migrated to the new **path-to-regexp v6**, which **no longer supports legacy wildcard routes**.
- The correct new syntax is:

```
app.get(/.*/, (req, res) => {
   res.sendFile(path.join(import.meta.dirname, "public", "index.html"));
});
```

- Patterns like /*path or /:path* throw errors because Express 5 requires named parameters for repetition.
- My final code using a **regex route** is the recommended solution:

```
app.get(/.*/, (req, res) => {
   res.sendFile(path.join(import.meta.dirname, "public", "index.html"));
});
```

• The AI confirmed that with Express 5, this version *is correct* and matches all non-API routes for SPA fallback routing.

⇒ none of them did work. found in documentation of express

https://expressjs.com/en/guide/migrating-5.html

3. Best error-handling pattern when using DB transactions

The AI explained:

- When performing multiple database writes inside a transaction, failure at ANY step must:
  - rollback immediately,
  - and propagate an error to the controller.
- The recommended pattern is:

```
try {
   db.exec("BEGIN");
```

```
    // ... multiple insert/update queries ...
    db.exec("COMMIT");
    return { success: true };
  } catch (err) {
    db.exec("ROLLBACK");
    throw err;   // let controller handle response formatting
  }
```

• The controller then has its own try/catch to send a proper HTTP status code:

```
try {
  const result = model.createArt(payload);
  res.status(201).json(result);
} catch (err) {
  res.status(500).json({ success: false, error: "Failed to create art" });
}
```

- This separation ensures:
  - **DB layer = business logic**
  - **Controller = API response logic**
- This is the industry-standard approach for safe transactions.

Summary of AI Assistance

The AI helped me:

- Understand when controllers should use try/catch and why GET endpoints don't always need them.
- Resolve the Express 5 wildcard routing issue by switching to a RegExp fallback route for the SPA.
- Implement safe transaction-based error handling where rollback always works correctly.

These clarifications allowed me to finalize a stable backend architecture.