**Lab – 3 Report**

**Submitted By: Khushpreet Kaur Lotey**

**1. Purpose of Each File in the Project**

**Backend Files (Located in backend/ Directory)**

* **server.js** (or **app.js**): The main entry point for the Express.js server. It initializes the app, sets up middleware (e.g., for JSON parsing and file uploads), defines routes, and starts the server on a configurable port (e.g., 3001). Purpose: To bootstrap the backend application and handle incoming HTTP requests.
* **routes/incidents.js**: Defines all REST API endpoints related to incidents (e.g., GET, POST, PATCH for incidents, and POST for bulk upload). Purpose: To organize and expose API logic for incident management, separating concerns from the main server file.
* **middleware/validation.js**: Contains validation functions for incident data (e.g., checking required fields, allowed values for category/severity). Purpose: To ensure data integrity by validating inputs before processing, preventing invalid data from entering the system.
* **data/store.js**: A simple module exporting an in-memory array to store incident objects. Purpose: To simulate a database for storing and retrieving incidents without persistence across restarts.
* **utils/csvParser.js**: Handles CSV file parsing and validation for bulk uploads. Purpose: To process uploaded CSV files, validate rows, and prepare data for insertion into the in-memory store.
* **package.json**: Configuration file listing dependencies (e.g., express, multer, csv-parse) and scripts. Purpose: To manage project dependencies and run commands like **npm start**.
* **README.md** (if included): Documentation for setup and usage. Purpose: To guide developers on installing and running the backend.

**Frontend Files (Located in frontend/ Directory)**

* **pages/index.js**: The home page of the Next.js app. Purpose: To serve as the landing page, potentially redirecting to the incident list or providing navigation links.
* **pages/incidents/index.js**: The incident list page. Purpose: To display a table/list of all incidents fetched from the backend, with links to individual details.
* **pages/incidents/create.js**: The create incident page. Purpose: To render a form for users to input new incident details and submit them to the backend.
* **pages/incidents/[id].js**: The dynamic incident details page. Purpose: To show full details of a specific incident and allow status updates.
* **pages/bulk-upload.js**: The CSV upload page. Purpose: To provide a file input for uploading CSV files and display upload results.
* **components/IncidentForm.js**: A reusable React component for forms (e.g., create or update). Purpose: To encapsulate form logic, validation, and rendering for reusability across pages.
* **components/IncidentTable.js**: A component for displaying incidents in a table. Purpose: To render incident data in a structured, readable format.
* **lib/api.js**: A utility module with functions to call backend APIs (e.g., using fetch). Purpose: To centralize API interactions, making them reusable and easier to maintain.
* **next.config.js**: Next.js configuration file. Purpose: To set environment variables (e.g., backend base URL) and customize the app.
* **package.json**: Lists frontend dependencies (e.g., next, react) and scripts. Purpose: To manage dependencies and run commands like **npm run dev**.
* **public/ Directory**: Contains static assets (e.g., images). Purpose: To serve files like icons or logos directly.
* **README.md**: Documentation for setup and usage. Purpose: To guide developers on installing and running the frontend.

**2. Detailed Explanation of Each Function in the Code**

This section explains key functions from the backend and frontend, including what they do, how they are used, and their purpose. Functions are grouped by file for clarity.

**Backend Functions**

* **In server.js**:
  + **app.listen(port, () => { ... })**: Starts the server on the specified port. Used: Called at the end of the file to launch the app. Purpose: To make the backend accessible for API requests.
  + **app.use(express.json())**: Middleware to parse JSON request bodies. Used: In server setup. Purpose: To enable handling of JSON data in POST/PATCH requests.
* **In routes/incidents.js**:
  + **router.get('/api/incidents', (req, res) => { ... })**: Retrieves all incidents from the in-memory store. Used: When the frontend fetches the incident list. Purpose: To expose incident data for viewing.
  + **router.get('/api/incidents/:id', (req, res) => { ... })**: Fetches a single incident by ID. Used: On the details page. Purpose: To provide detailed incident information.
  + **router.post('/api/incidents', (req, res) => { ... })**: Creates a new incident after validation. Used: When submitting the create form. Purpose: To add new incidents to the system.
  + **router.patch('/api/incidents/:id/status', (req, res) => { ... })**: Updates an incident's status with validation. Used: On the details page for status changes. Purpose: To modify incident status while enforcing rules.
  + **router.post('/api/incidents/bulk-upload', (req, res) => { ... })**: Handles CSV upload, parses, validates, and creates incidents. Used: On the bulk-upload page. Purpose: To enable batch incident creation from files.
* **In middleware/validation.js**:
  + **validateIncident(data)**: Checks fields like title length, category values, etc. Used: In POST and bulk-upload routes. Purpose: To ensure data quality and return errors for invalid inputs.
* **In utils/csvParser.js**:
  + **parseCSV(file)**: Parses the CSV and validates rows. Used: In the bulk-upload endpoint. Purpose: To convert file data into usable objects for the store.

**Frontend Functions**

* **In pages/incidents/index.js**:
  + **useEffect(() => { fetchIncidents(); }, [])**: Fetches incidents on page load. Used: To populate the list. Purpose: To display current incidents dynamically.
  + **fetchIncidents()**: Calls the API and updates state. Used: In useEffect. Purpose: To retrieve and render incident data.
* **In pages/incidents/create.js**:
  + **handleSubmit(e)**: Validates form and sends POST request. Used: On form submission. Purpose: To create incidents and handle success/error responses.
  + **useState** hooks (e.g., for title, description): Manage form state. Used: To track user inputs. Purpose: To enable reactive form updates.
* **In pages/incidents/[id].js**:
  + **updateStatus(newStatus)**: Sends PATCH request to update status. Used: On dropdown change. Purpose: To modify incident status via API.
* **In pages/bulk-upload.js**:
  + **handleUpload(file)**: Submits the file via POST. Used: On upload button click. Purpose: To process CSV and display results.
* **In lib/api.js**:
  + **fetchIncidents()**: Wrapper for GET /api/incidents. Used: Across pages. Purpose: To abstract API calls for consistency.
  + **createIncident(data)**: Wrapper for POST /api/incidents. Used: In create page. Purpose: To handle creation logic centrally.
* **In components/IncidentForm.js**:
  + **render()**: Returns JSX for form fields. Used: In create/update pages. Purpose: To provide a reusable UI for data entry.

**3. Application and Data Workflows**

**Overall Application Workflow**

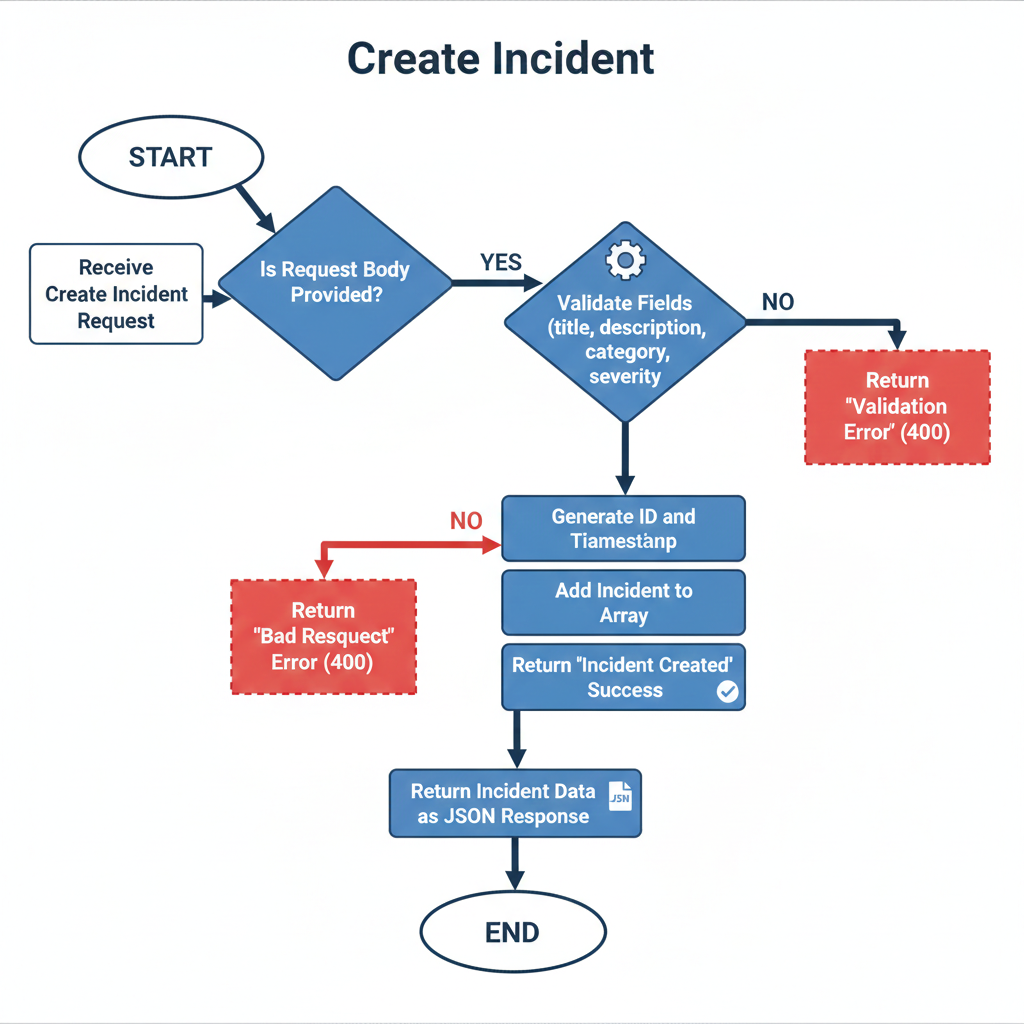
1. **User Interaction**: Users access the Next.js frontend (e.g., via browser). Pages like **/incidents** trigger API calls to the Express backend.
2. **Frontend Processing**: React components handle state (e.g., form inputs) and use **fetch** to send requests to the backend (e.g., via environment variable **NEXT\_PUBLIC\_API\_URL**).
3. **Backend Processing**: Express routes validate inputs, interact with the in-memory store (array), and return JSON responses.
4. **Data Flow**: Data flows unidirectionally from frontend (user input) → backend (validation/storage) → frontend (display). Errors are handled by returning HTTP codes and messages.
5. **Key Integrations**: Frontend uses **fetch** for API calls; backend uses middleware for parsing. No persistence—data resets on restart.
6. **Error Handling**: Frontend displays alerts for API errors; backend returns 400/404 with JSON.

**Data Workflows for Core Operations**

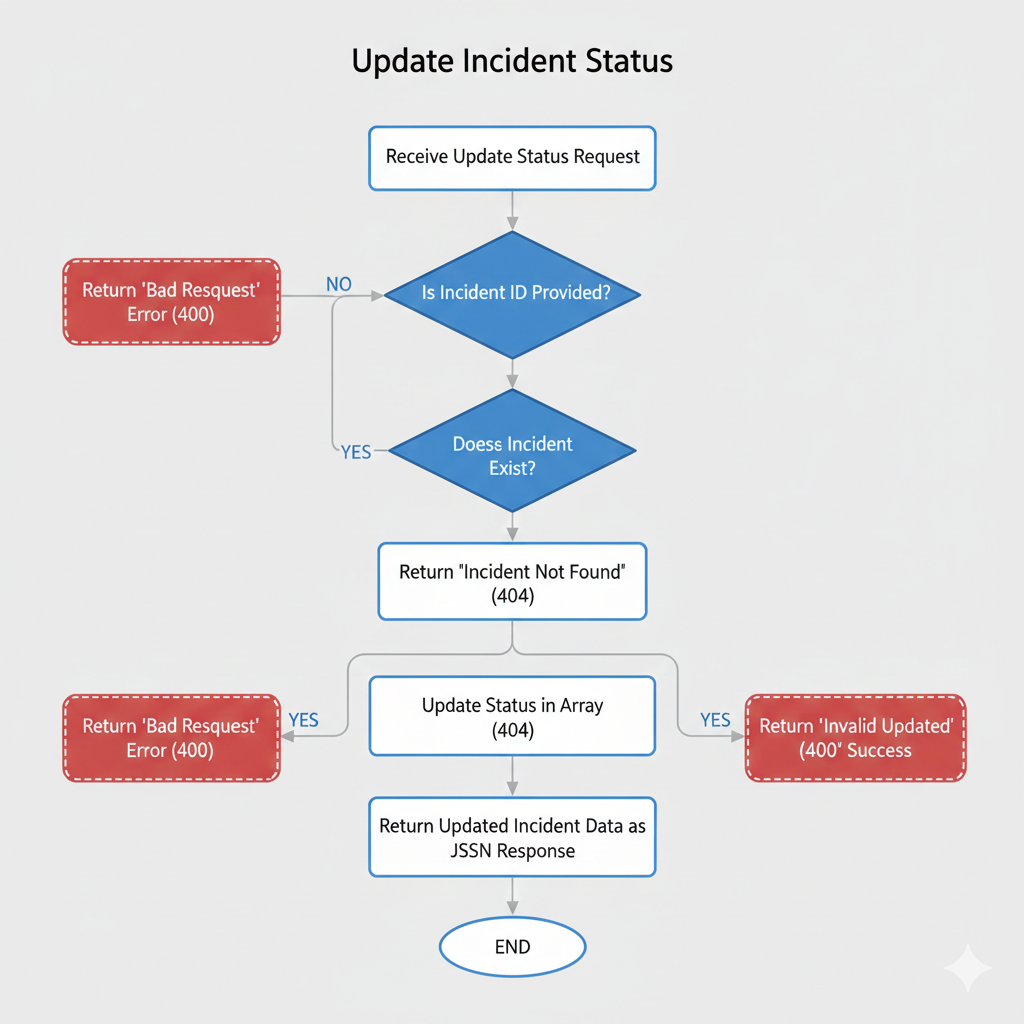
* **Create Incident**: User fills form → Frontend validates → POST to backend → Backend validates → Adds to array → Returns success → Frontend redirects.
* **Read Incident**: User clicks link → Frontend fetches by ID → Backend retrieves from array → Returns data → Frontend renders details.
* **Update Incident**: User selects status → Frontend sends PATCH → Backend checks transition → Updates array → Returns updated object → Frontend refreshes.
* **Bulk Upload**: User selects CSV → Frontend uploads file → Backend parses/validates → Creates valid incidents → Returns summary → Frontend shows results.

**4. Flowcharts**

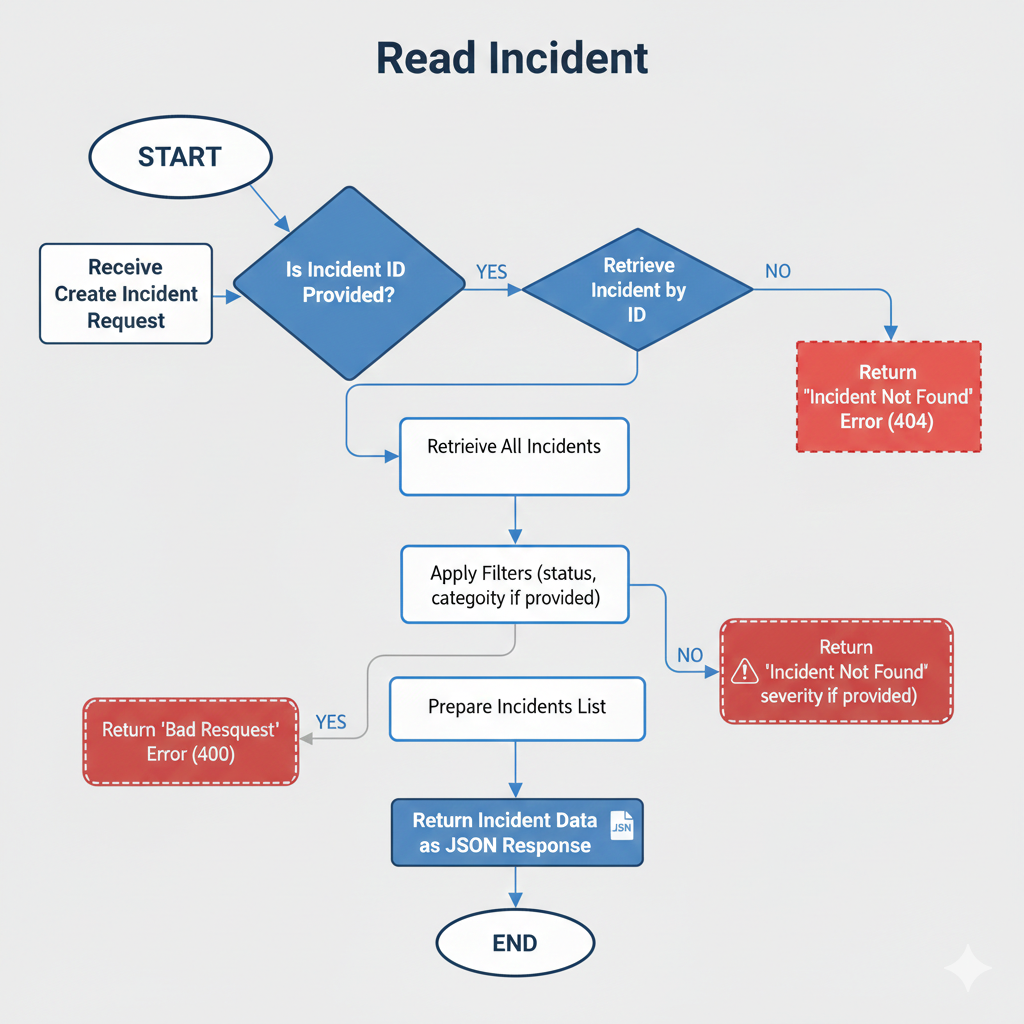
**Flowchart 1: Create Incident**



**Flowchart 2: Update Incident**



**Flowchart 3: Read Incident**



**Flowchart 4: Bulk Upload**

