Final Report

a. Project Overview:

Title: Jadwill – A Smart Tourist Planning Platform

Jadwill is an online platform that aims to increase Saudi domestic tourism through facilitating local guides and experienced suppliers to create and manage customized activity schedules. The platform brings together tourists, experienced providers, and guides to deliver authentic and experiential tourism experiences. Each schedule — a day or longer — is created by professionals, locals, who include key sights, hidden gems, and local customs. With Jadwill, visitors get an opportunity to discover selected experiences made up of the local residents' perspective, where they can actually live it as if they are there rather than just driving by. The initiative supports cultural exchange, stimulates the economy through local industry support, and advances the practice of sustainable home-based tourism.

b. Technologies and Tools:

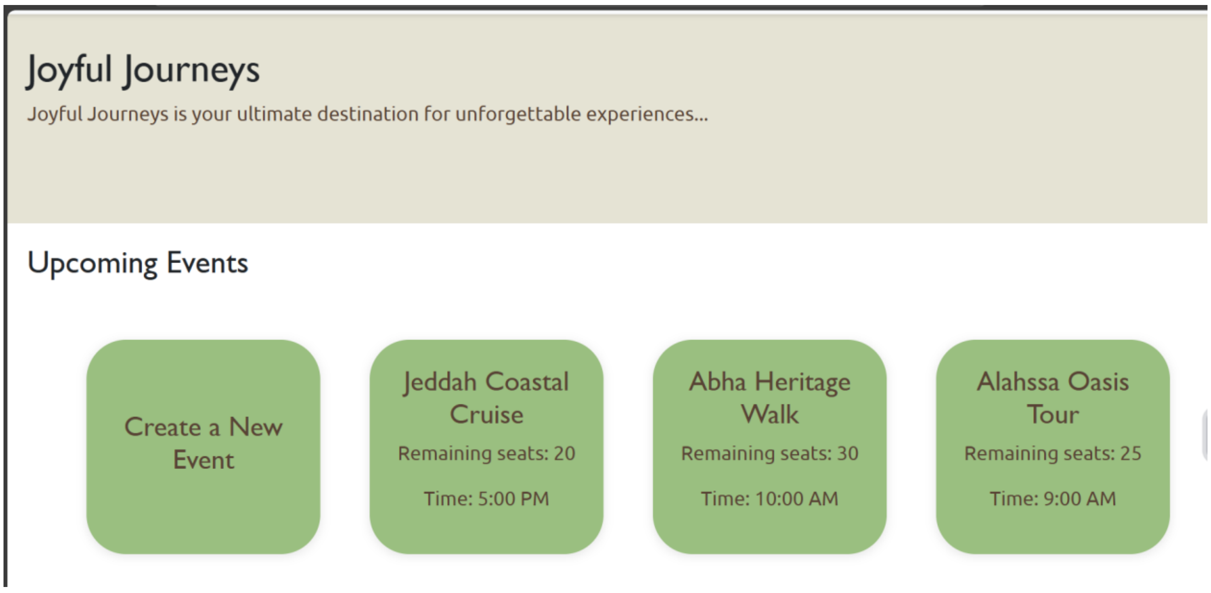
- **CSS**
  Used to build and style the frontend layout and user interface.
- **JavaScript**
  Core programming language used for both frontend interactivity and backend logic.
- **React.js**
  A JavaScript library used to create dynamic, component-based user interfaces.
- **Node.js**
  A runtime environment used to build the backend server and handle requests.
- **Express.js**
  A web application framework for Node.js used to create RESTful APIs.
- **MongoDB Atlas**
  A cloud-hosted NoSQL database used to store users, activities, reservations, and reviews.
- **Bootstrap**
  A frontend framework used for responsive layout, components, and grid system.

- **Postman**
  Used to test API endpoints during development and ensure backend functionality.
- **Figma**
  UI/UX design tool used to create and share mockups and design prototypes.
- **Git & GitHub**
  Used for version control and collaborative development among team members.
- **Visual Studio Code**
  Code editor used by the development team for writing and testing code.

c. GUI Screenshots:

Activity Provider pages:
The main dashboard interfac**e** for Activity Providers, where upcoming events are displayed in a clean, cards.

- Each event card includes key information: event title, remaining seats, and start time, making it easy for guides to quickly assess upcoming activities. Also if he clicks the card more details will be displayed
- A "Create a New Event" card allows guides to easily navigate to the event creation form.
- The layout uses a responsive grid with rounded, color-coded cards to emphasize a friendly, engaging



Joyful Journeys

Joyful Journeys is your ultimate destination for unforgettable experiences...

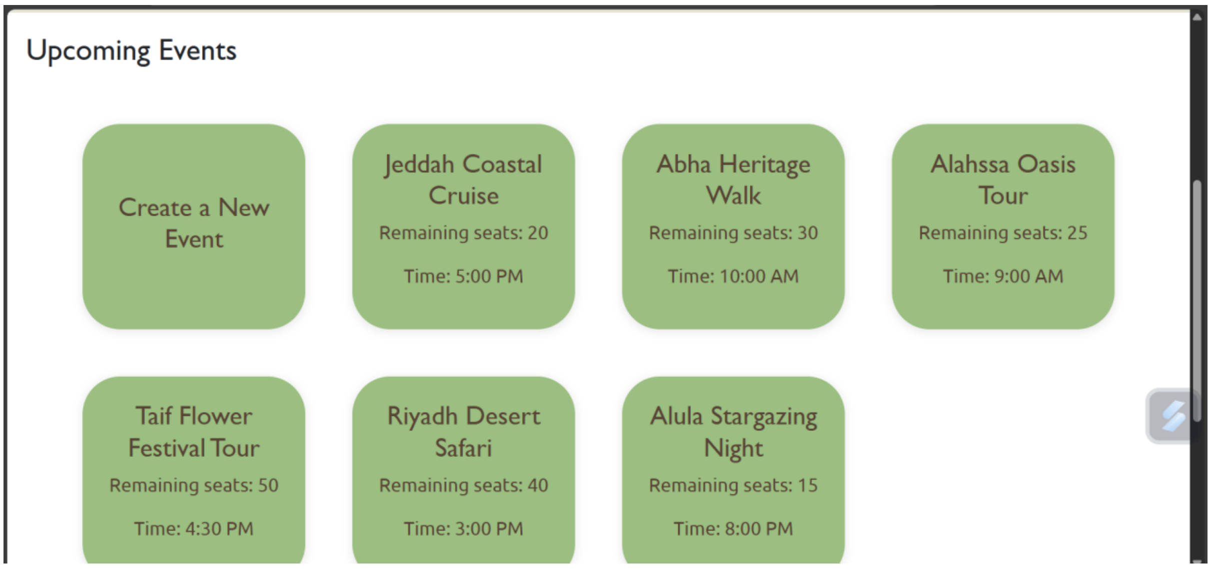
Upcoming Events

Create a New Event

Jeddah Coastal Cruise
Remaining seats: 20
Time: 5:00 PM

Abha Heritage Walk
Remaining seats: 30
Time: 10:00 AM

Alahssa Oasis Tour
Remaining seats: 25
Time: 9:00 AM



Upcoming Events

Create a New Event

Jeddah Coastal Cruise
Remaining seats: 20
Time: 5:00 PM

Abha Heritage Walk
Remaining seats: 30
Time: 10:00 AM

Alahssa Oasis Tour
Remaining seats: 25
Time: 9:00 AM

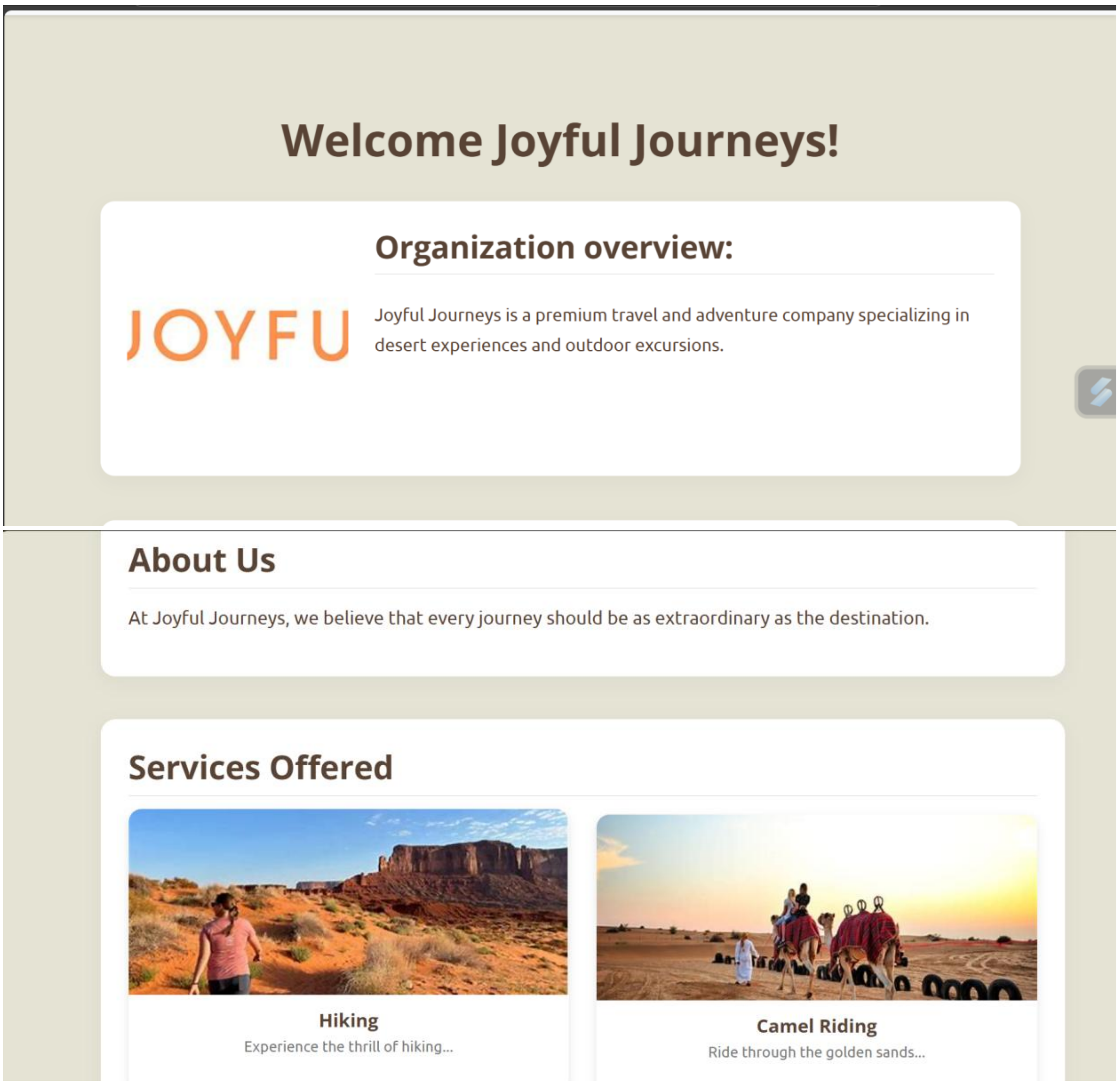
Taif Flower Festival Tour
Remaining seats: 50
Time: 4:30 PM

Riyadh Desert Safari
Remaining seats: 40
Time: 3:00 PM

Alula Stargazing Night
Remaining seats: 15
Time: 8:00 PM

The profile page for an activity provider (Joyful Journeys). The page is organized into distinct sections for better usability:

- Displays the organization’s name and a brief company description, offering a clear identity to tourists.
- Introduces the highlights available services using illustrated cards (Hiking, Camel Riding, etc).
- Lists essential contact details such as email and phone numbers for easy communication.
- Allows the provider to update their profile content, ensuring flexibility and control.



Welcome Joyful Journeys!

Organization overview:

JOYFU

Joyful Journeys is a premium travel and adventure company specializing in desert experiences and outdoor excursions.

About Us

At Joyful Journeys, we believe that every journey should be as extraordinary as the destination.

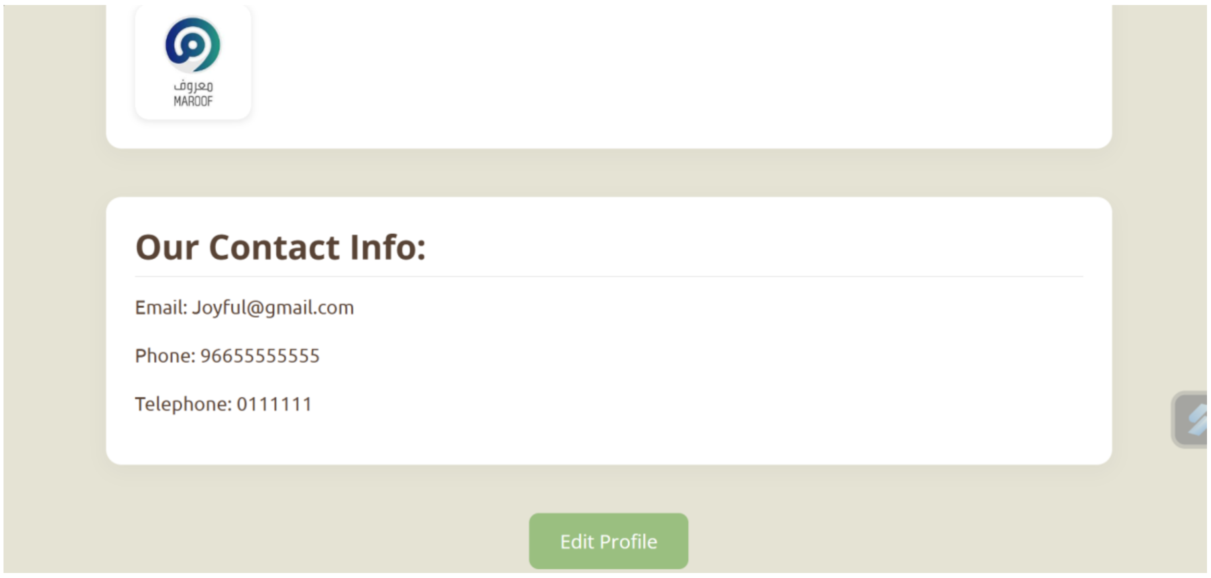
Services Offered

Hiking

Experience the thrill of hiking...

Camel Riding

Ride through the golden sands...



Our Contact Info:

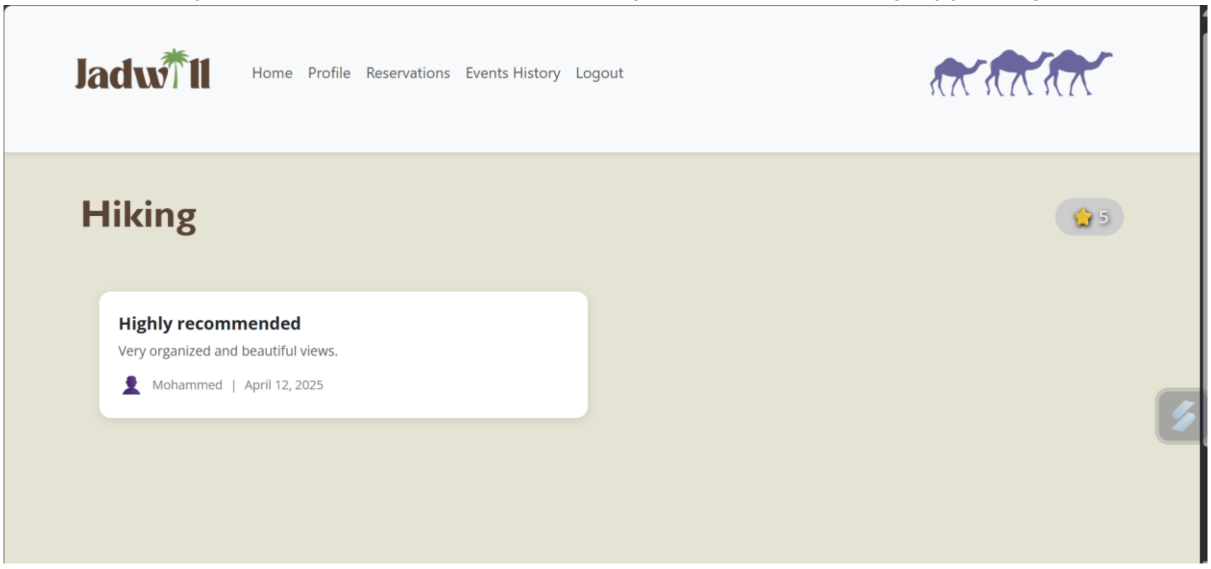
Email: Joyful@gmail.com

Phone: 96655555555

Telephone: 0111111

Detailed view of an activity rating (*Hiking*) along with user-generated feedback and a rating.

- Providers can read reviews submitted by other users. Each review includes a title, comment, reviewer name, and submission date.
- A visual star icon and number (★ 5) represent the average rating for the activity, enhancing transparency and trust.
- The minimal layout ensures that reviews are easy to read and visually appealing.



Hiking

Highly recommended

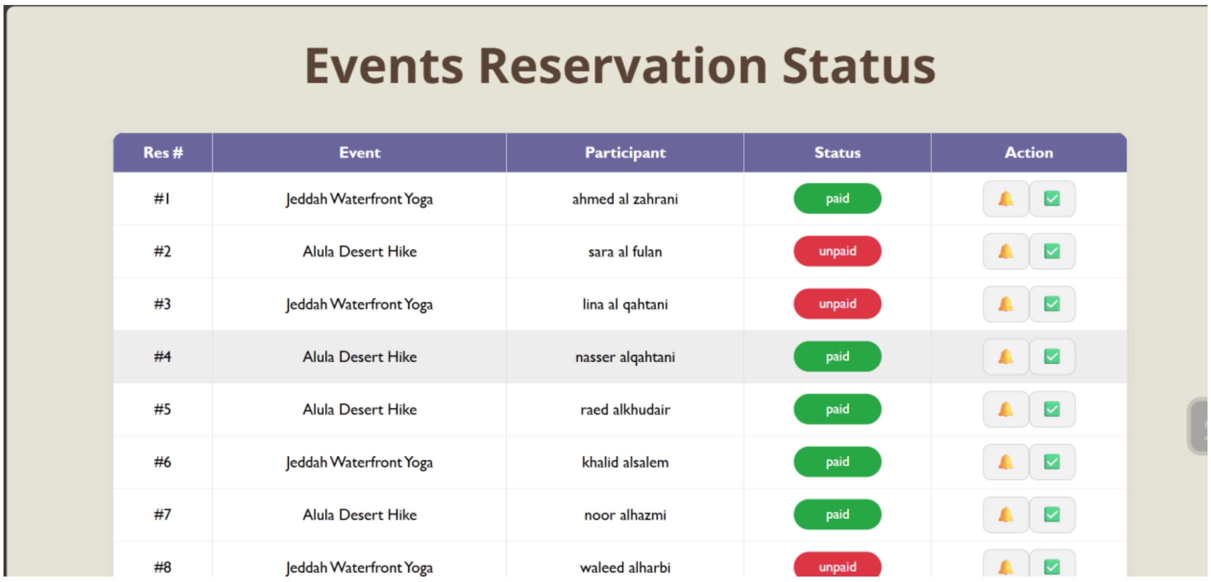
Very organized and beautiful views.

Mohammed | April 12, 2025

Activity providers view and manage reservations for their events. Each row in the table presents:
Reservation Number, Event Name, and Participant information
Payment Status badges (green for "paid", red for "unpaid")
Action Buttons:

    Alert icon for notifications or reminders
    Confirmation checkbox to mark completion or attendance

The clean tabular layout helps providers monitor participation and follow up on unpaid bookings efficiently. This screen supports organized event handling and improves provider control over event



Events Reservation Status

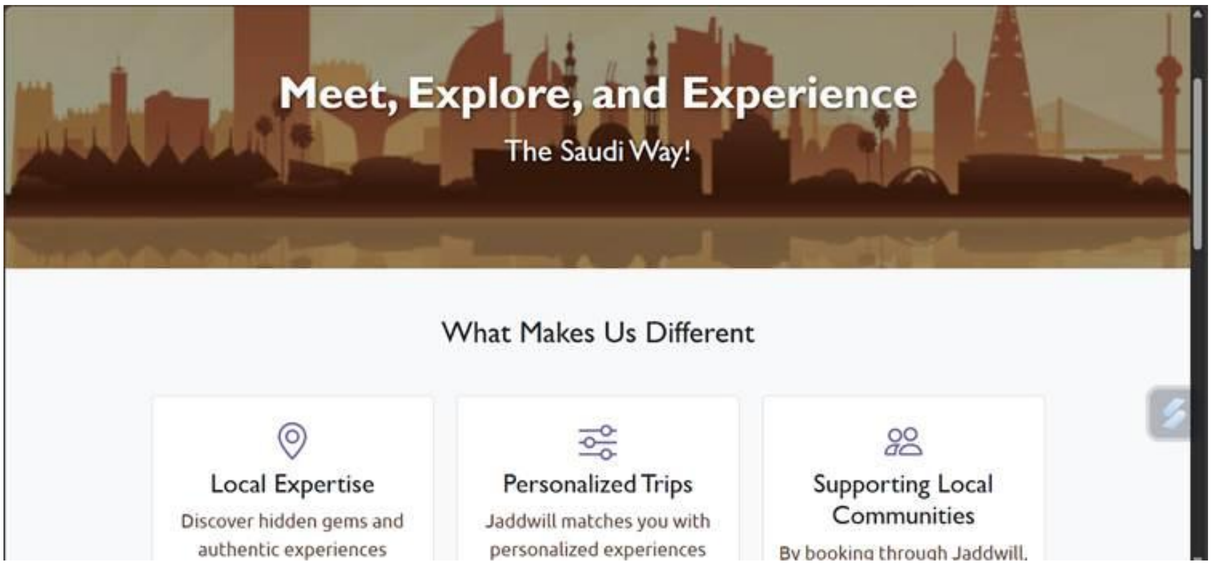
Res #	Event	Participant	Status	Action
#1	Jeddah Waterfront Yoga	ahmed al zahrani	paid	🔔 ✅
#2	Alula Desert Hike	sara al fulan	unpaid	🔔 ✅
#3	Jeddah Waterfront Yoga	lina al qahtani	unpaid	🔔 ✅
#4	Alula Desert Hike	nasser alqahtani	paid	🔔 ✅
#5	Alula Desert Hike	raed alkhudair	paid	🔔 ✅
#6	Jeddah Waterfront Yoga	khalid alsalem	paid	🔔 ✅
#7	Alula Desert Hike	noor alhazmi	paid	🔔 ✅
#8	Jeddah Waterfront Yoga	waleed alharbi	unpaid	🔔 ✅

Tourist Pages:
The landing page provides a welcoming and informative introduction to the platform. It features a visually engaging hero section with the tagline.

"What Makes Us Different" section highlights key value propositions:

- Local Expertise: Emphasizes authentic, insider-led experiences.
- Personalized Trips: Promotes recommendations for users.
- Supporting Local Communities: Reinforces the platform’s commitment to local impact.

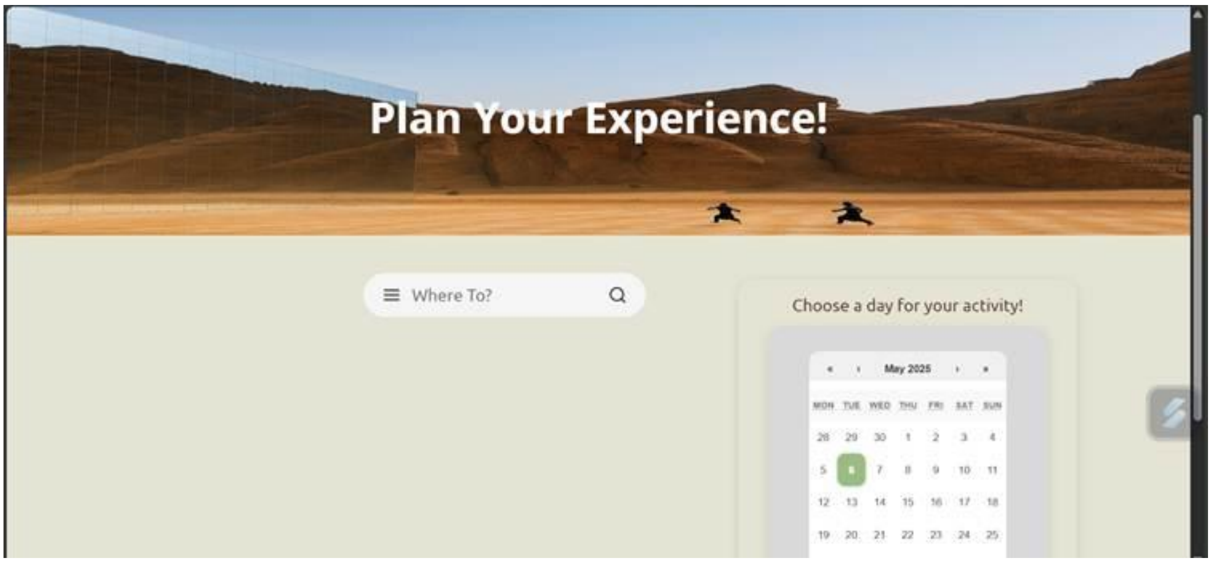
This pages combines strong branding with concise messaging, ensuring a compelling first impression for visitors and effectively communicating the platform’s mission

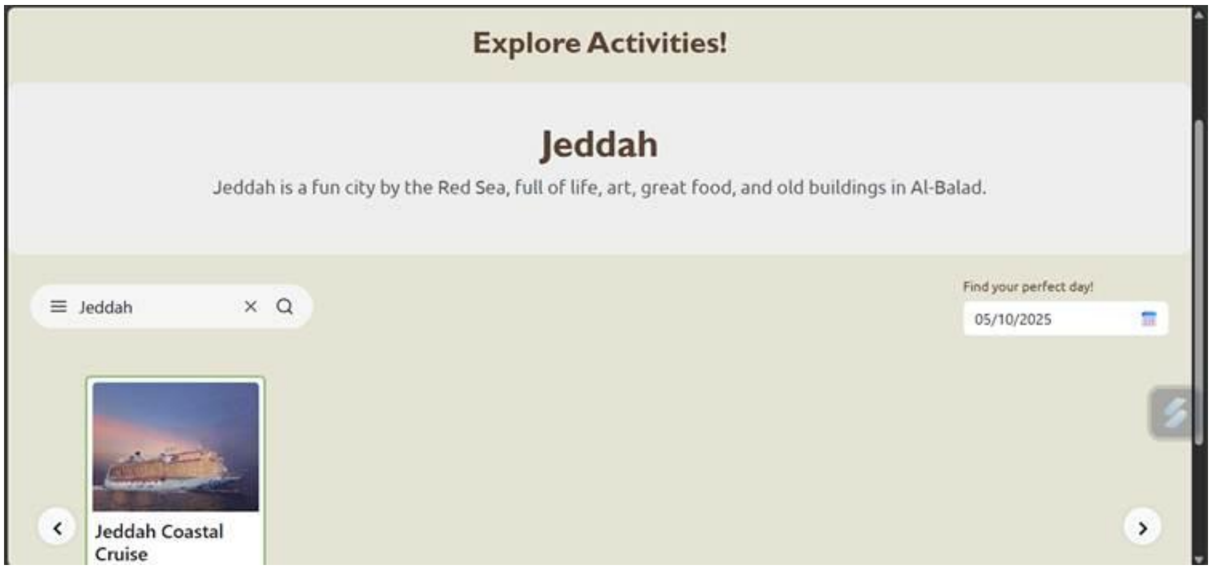

Meet, Explore, and Experience
The Saudi Way!
What Makes Us Different
Local Expertise
Discover hidden gems and authentic experiences
Personalized Trips
Jaddwill matches you with personalized experiences
Supporting Local Communities
By booking through Jaddwill.

This interface allows tourists to explore and book activities by city and date:

- Plan Your Experience: Features a scenic background with a destination search bar and an interactive calendar for selecting dates.
- Explore Activities by City: Once a city and date are selected, tourists are shown available activities like the *Jeddah Coastal Cruise* with visual previews, enhancing discovery and engagement.
- Dynamic Filtering: The dropdown and calendar components allow seamless filtering to personalize trip planning.

The layout is intuitive and visually engaging, helping users find and plan activities that match their interests and schedules




Explore Activities!

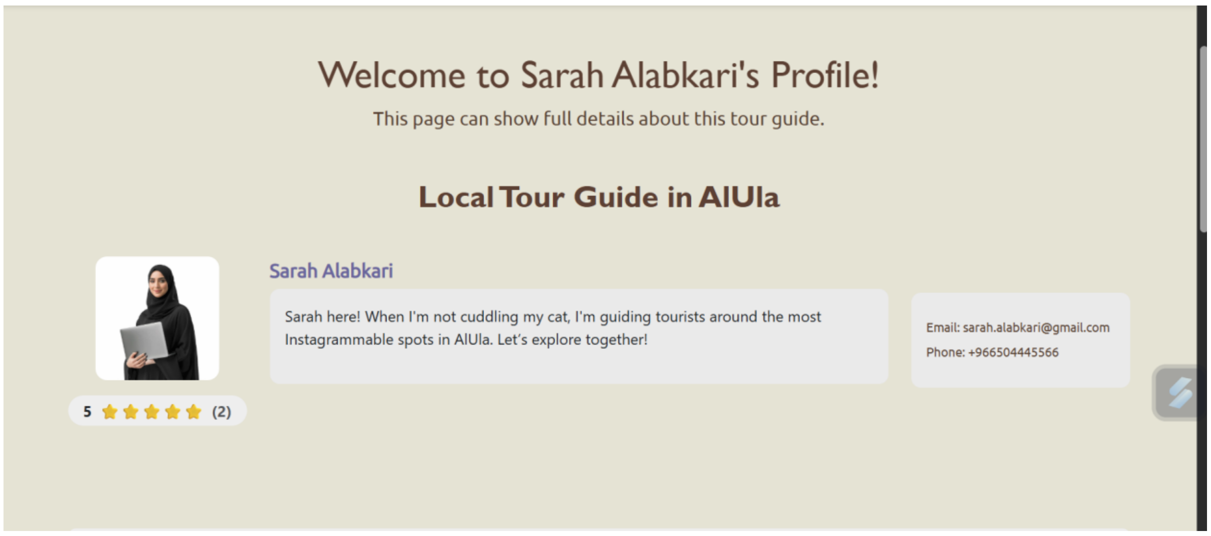
Jeddah

Jeddah is a fun city by the Red Sea, full of life, art, great food, and old buildings in Al-Balad.

A profile of a tour guide, enhancing trust and personalization for tourists.

- Displays the guide’s name, bio, and region of expertise (AlUla).
- Includes the guide’s photo, email, and phone number for easy communication.
- Shows the average rating and number of reviews, helping tourists assess the guide's reputation.

The friendly design and detailed information create a more personal experience, encouraging tourists to connect confidently with local guides.


Welcome to Sarah Alabkari's Profile!

This page can show full details about this tour guide.

Local Tour Guide in AlUla

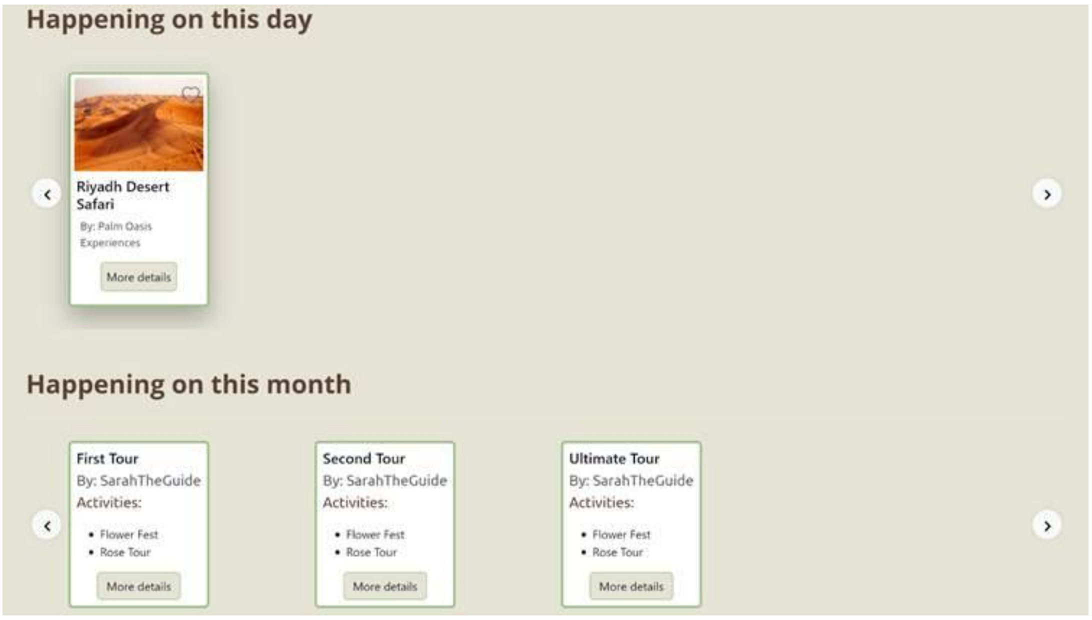
Sarah Alabkari

Sarah here! When I'm not cuddling my cat, I'm guiding tourists around the most Instagrammable spots in AlUla. Let’s explore together!

Email: sarah.alabkari@gmail.com
Phone: +966504445566

Tour Guide pages:
The tour guide dashboard offers a comprehensive view of performance, engagement, and upcoming tours.

- Displays count for completed, canceled, and scheduled tours, along with total attendees over a selected date range.
- Visual bar chart showing monthly earnings trends for the selected year.
- Highlights the most attended tours, offering insights into the tour popularity.
- Shows feedback and ratings from past tourists, helping guides improve service quality.
- Sections for activities scheduled on the selected day or throughout the month keep guides informed of their schedule.



Happening on this day

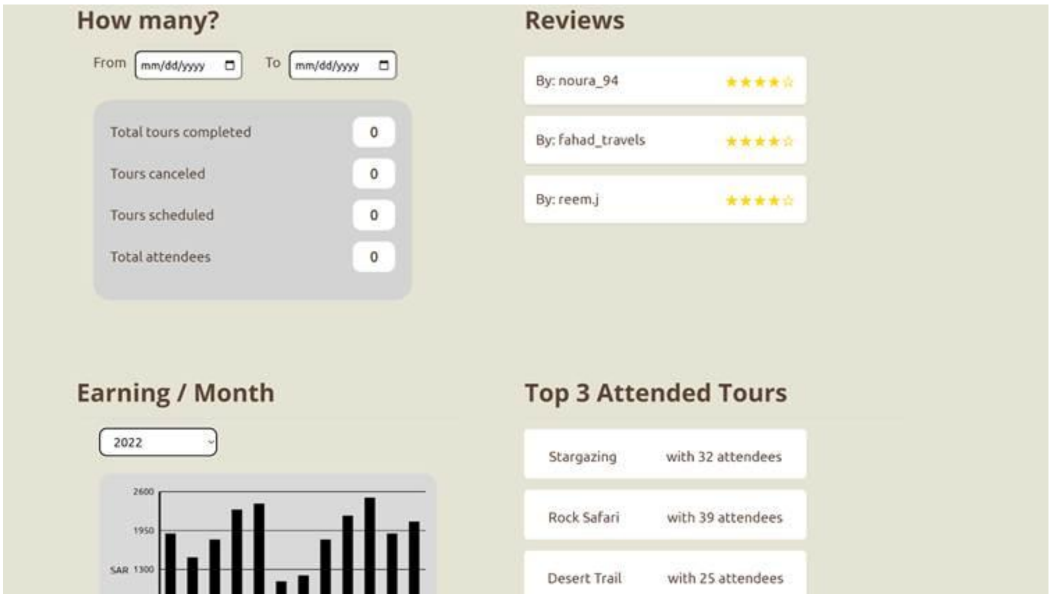
Riyadh Desert Safari
By: Palm Oasis Experiences
More details

Happening on this month

First Tour
By: SarahTheGuide
Activities:
- Flower Fest
- Rose Tour
More details

Second Tour
By: SarahTheGuide
Activities:
- Flower Fest
- Rose Tour
More details

Ultimate Tour
By: SarahTheGuide
Activities:
- Flower Fest
- Rose Tour
More details



How many?

From mm/dd/yyyy To mm/dd/yyyy

Total tours completed	0
Tours canceled	0
Tours scheduled	0
Total attendees	0

Reviews

By: noura_94	★★★★☆
By: fahad_travels	★★★★☆
By: reem.j	★★★★☆

Earning / Month

2022

Top 3 Attended Tours

Stargazing	with 32 attendees
Rock Safari	with 39 attendees
Desert Trail	with 25 attendees

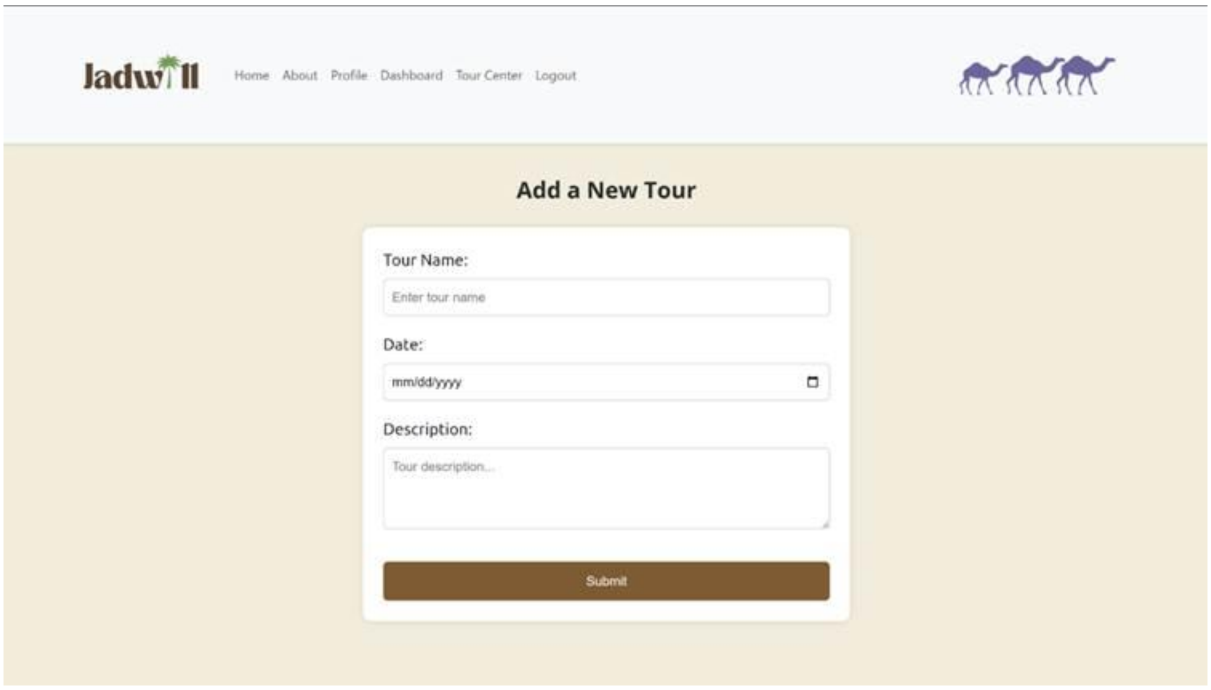
These two screens empower tour guides to manage their tours directly and efficiently from the web interface.

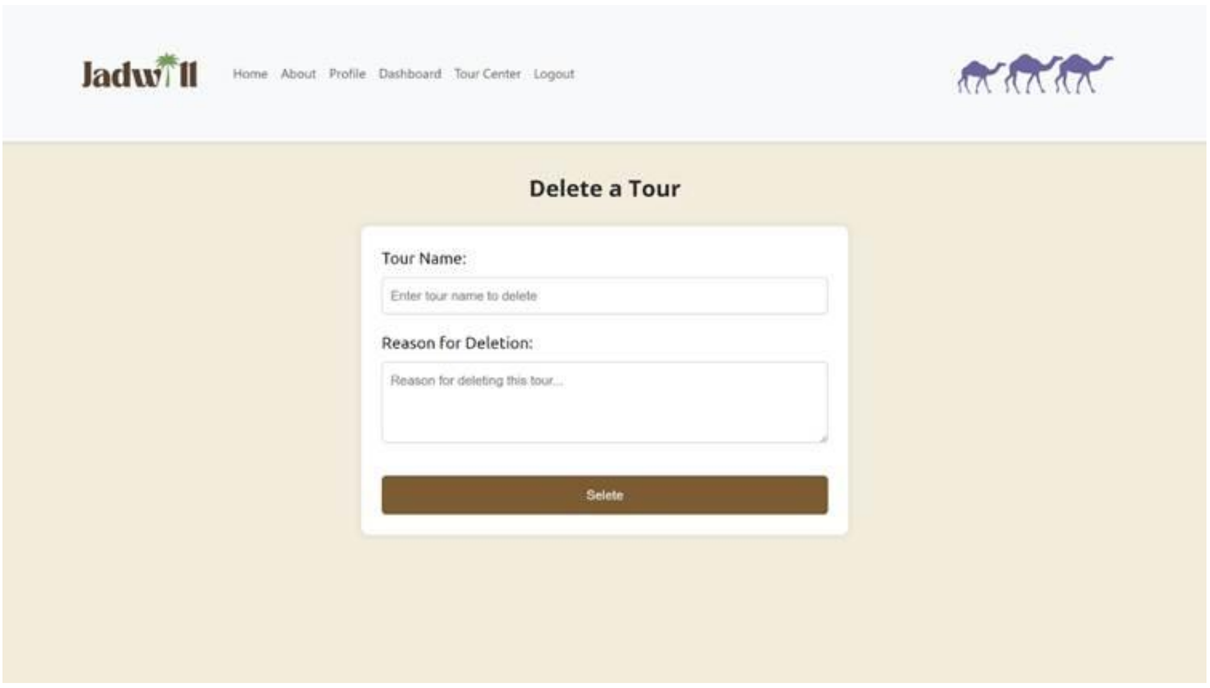
Add a Tour

- Tour Name, Date, Description Fields: Enables the guide to create a new tour by specifying its name, scheduled date, and description.
- Submit Button: Sends the entered details to the backend to create and store the new tour in the database.
  Purpose: Helps guides expand their offerings and keep their profile active with fresh events.

Delete a Tour

- Tour Name Field: Specifies the tour to be deleted.
- Reason for Deletion Field: Encourages input on why the tour is being removed—for admin awareness or audit purposes.
- Delete Button: Sends a delete request to the backend to remove the selected tour permanently.





Delete a Tour

Tour Name:

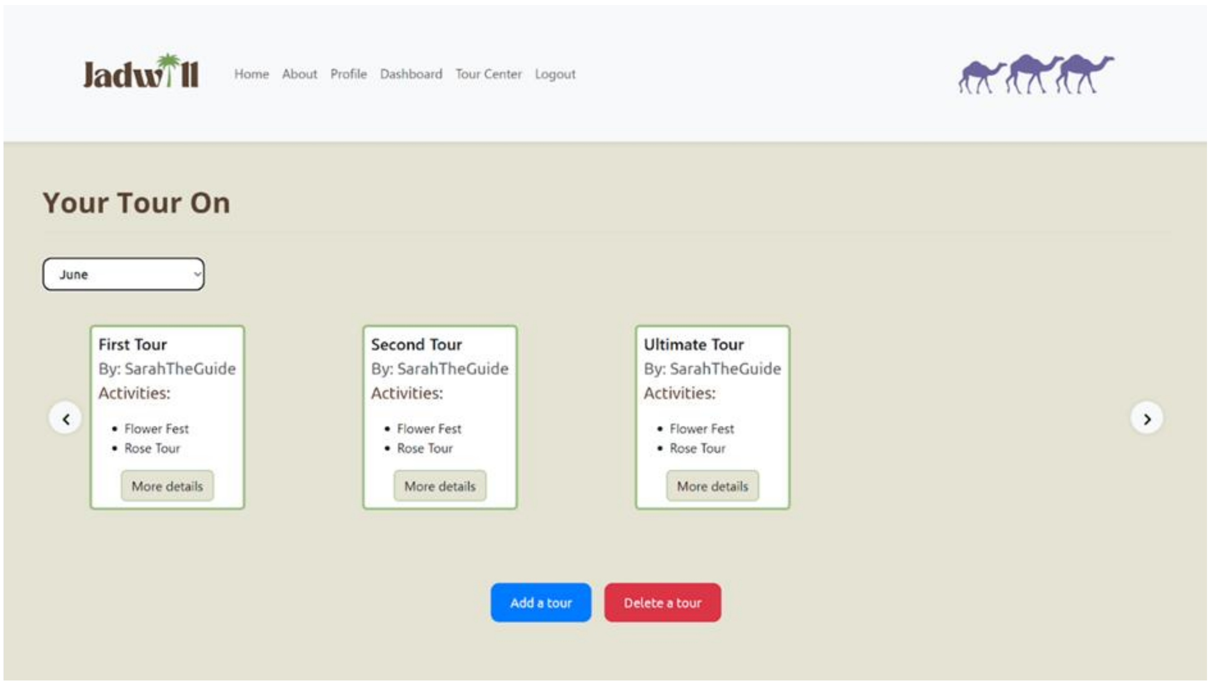
Enter tour name to delete

Reason for Deletion:

Reason for deleting this tour...

This interface allows tour guides to view, manage, and interact with their scheduled tours on a monthly basis.

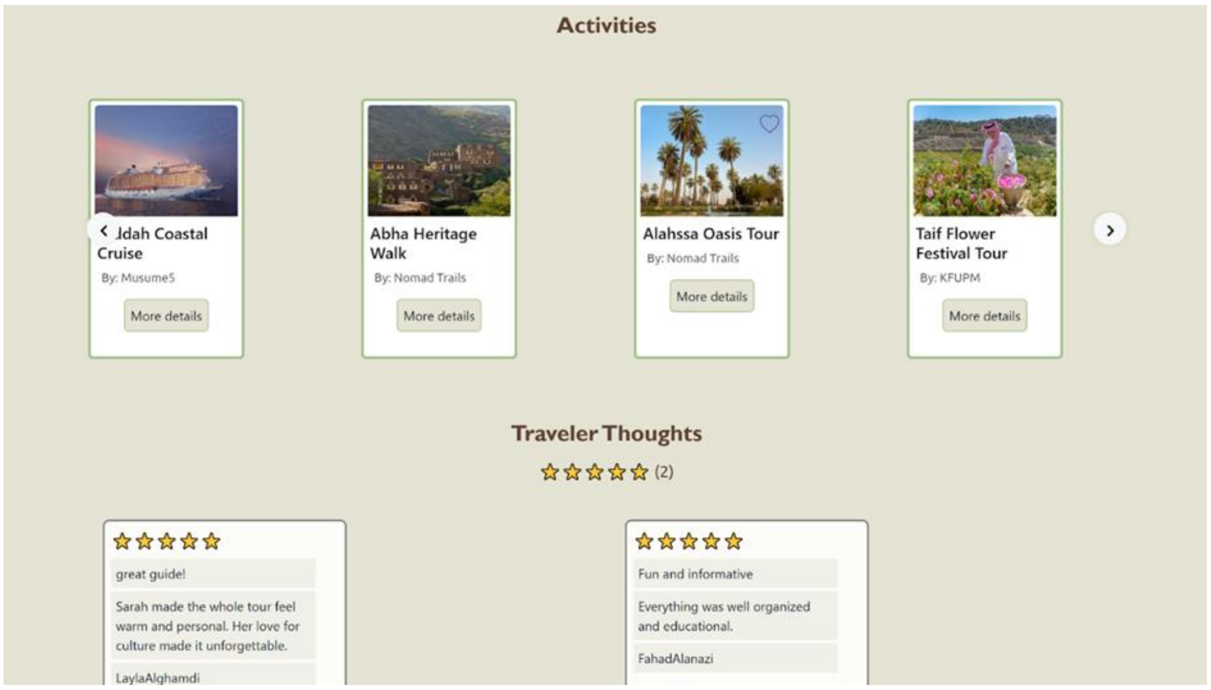
- Allows filtering tours by month to view only relevant ones.
- Each card displays the tour title, guide name, and list of included activities.
- Users can scroll left or right to browse more tours.
- Opens a detailed view of the selected tour.
- Redirects to the tour creation form for adding new events.
- Opens the deletion form to remove a selected tour



Activities Section:

- Guides can see all the activities they offer, each displayed in a card format.
- Easily scroll through tours using side buttons.
- "More Details" Button: Provides quick access to individual tour information.
- Highlights the average rating based on tourist feedback
- Tourists' comments with reviewer names give guides insight into performance and satisfaction.

This screen empowers guides to track engagement with their tours and gather direct feedback, helping them improve and promote their services effectively.



Activities

‹ ddah Coastal Cruise	Abha Heritage Walk	Alahssa Oasis Tour	Taif Flower Festival Tour
By: Musume5	By: Nomad Trails	By: Nomad Trails	By: KFUPM
More details	More details	More details	More details

Traveler Thoughts

☆☆☆☆☆ (2)

☆☆☆☆☆
great guide!
Sarah made the whole tour feel warm and personal. Her love for culture made it unforgettable.
LaylaAlghamdi

☆☆☆☆☆
Fun and informative
Everything was well organized and educational.
FahadAlanazi

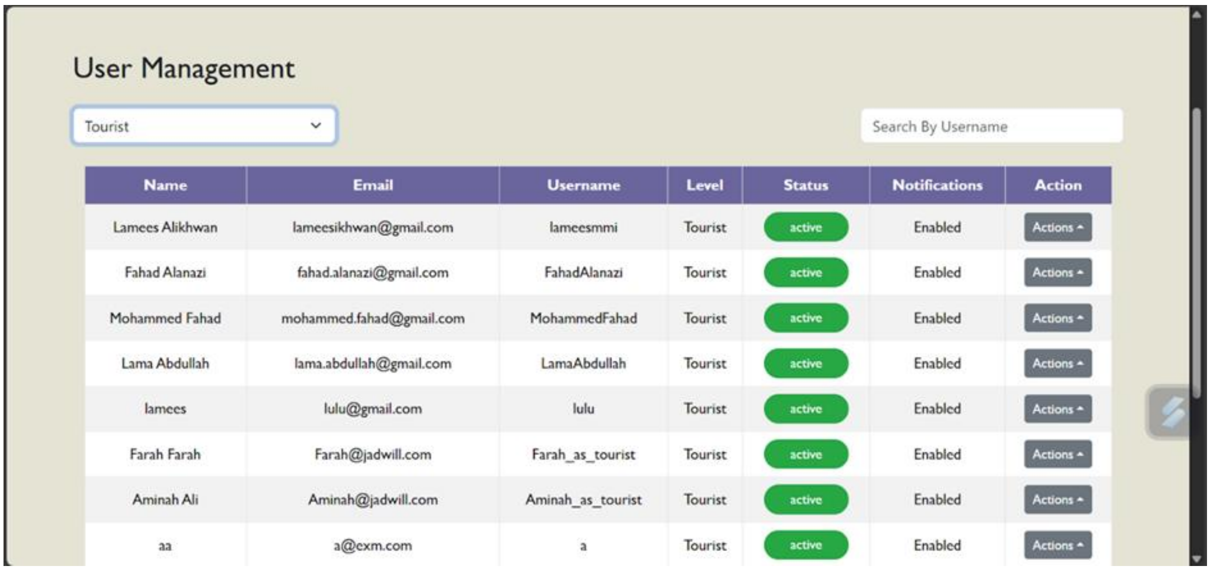
Admin pages:

User Management Page

Manage all users of the platform by role.

- Filter by role (Tourist, Guide, etc.).
- Search users by username.
- Status and notification indicators.
- Action dropdown for enabling/disabling users or updating roles.

Focus on visibility and user status control with toggles for efficient user administration.



User Management

Tourist

Search By Username

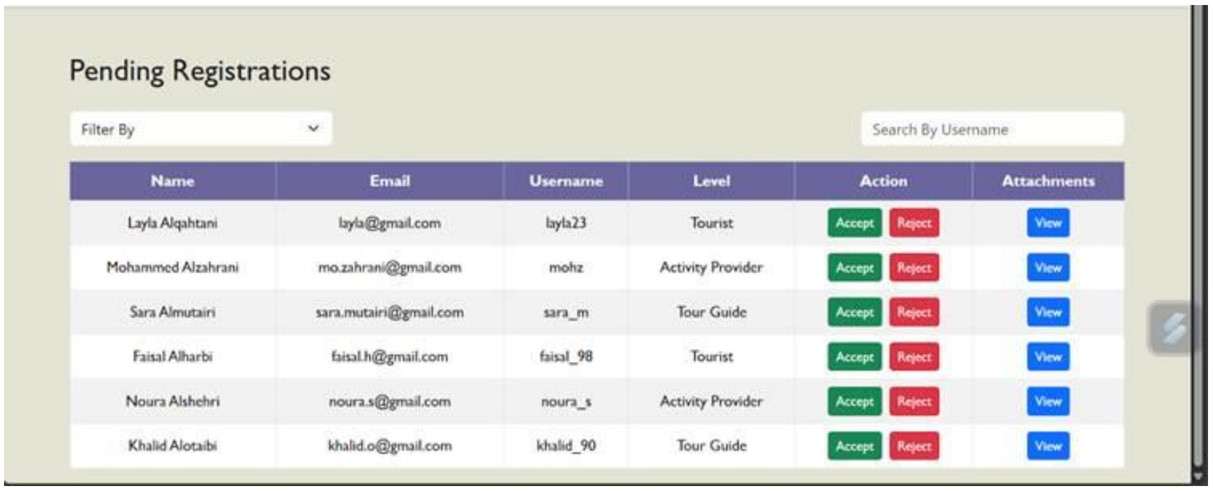
Name	Email	Username	Level	Status	Notifications	Action
Lamees Alikhwan	lameesikhwan@gmail.com	lameesmmi	Tourist	active	Enabled	Actions
Fahad Alanazi	fahad.alanazi@gmail.com	FahadAlanazi	Tourist	active	Enabled	Actions
Mohammed Fahad	mohammed.fahad@gmail.com	MohammedFahad	Tourist	active	Enabled	Actions
Lama Abdullah	lama.abdullah@gmail.com	LamaAbdullah	Tourist	active	Enabled	Actions
lamees	lulu@gmail.com	lulu	Tourist	active	Enabled	Actions
Farah Farah	Farah@jadwill.com	Farah_as_tourist	Tourist	active	Enabled	Actions
Aminah Ali	Aminah@jadwill.com	Aminah_as_tourist	Tourist	active	Enabled	Actions
aa	a@exm.com	a	Tourist	active	Enabled	Actions

Pending Registrations Page

Enables admin to approve or reject newly registered users.

- Shows user info: Name, Email, Username, User Level.
- Two action buttons per row: Accept (green) and Reject (red).
- Attachment viewer for documents/images.

Organized table with immediate control buttons to streamline account verification.



Pending Registrations

Filter By

Search By Username

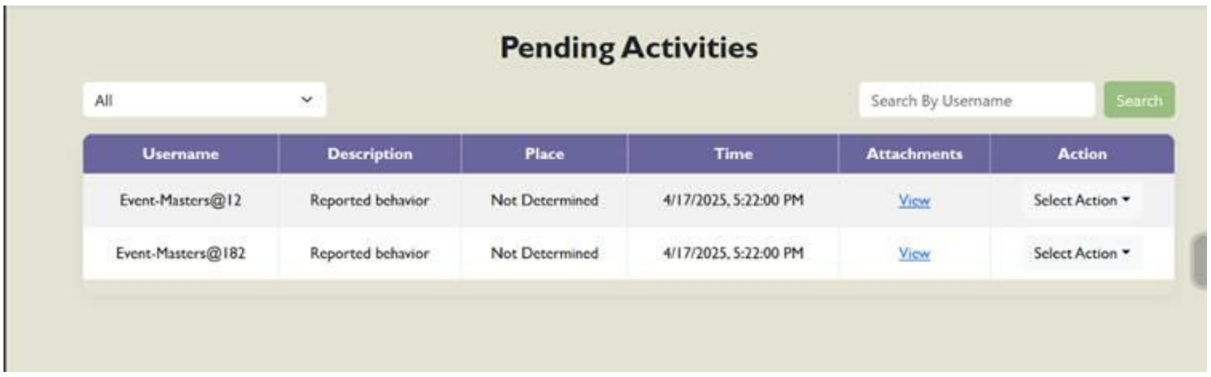
Name	Email	Username	Level	Action	Attachments
Layla Alqahtani	layla@gmail.com	layla23	Tourist	Accept Reject	View
Mohammed Alzahrani	mo.zahrani@gmail.com	mohz	Activity Provider	Accept Reject	View
Sara Almutairi	sara.mutairi@gmail.com	sara_m	Tour Guide	Accept Reject	View
Faisal Alharbi	faisal.h@gmail.com	faisal_98	Tourist	Accept Reject	View
Noura Alshehri	noura.s@gmail.com	noura_s	Activity Provider	Accept Reject	View
Khalid Alotaibi	khalid.o@gmail.com	khalid_90	Tour Guide	Accept Reject	View

Pending Activities Page

Allows the admin to review user-submitted activities that are awaiting approval.

- Dropdown filter (top-left) to filter by category.
- Search bar to find by username.
- Table view showing: Username, Description, Place, Time, Attachments (View), and Action (Approve/Reject).

Table layout with responsive action controls simplifies moderation.



Pending Activities

All

Search By Username

Search

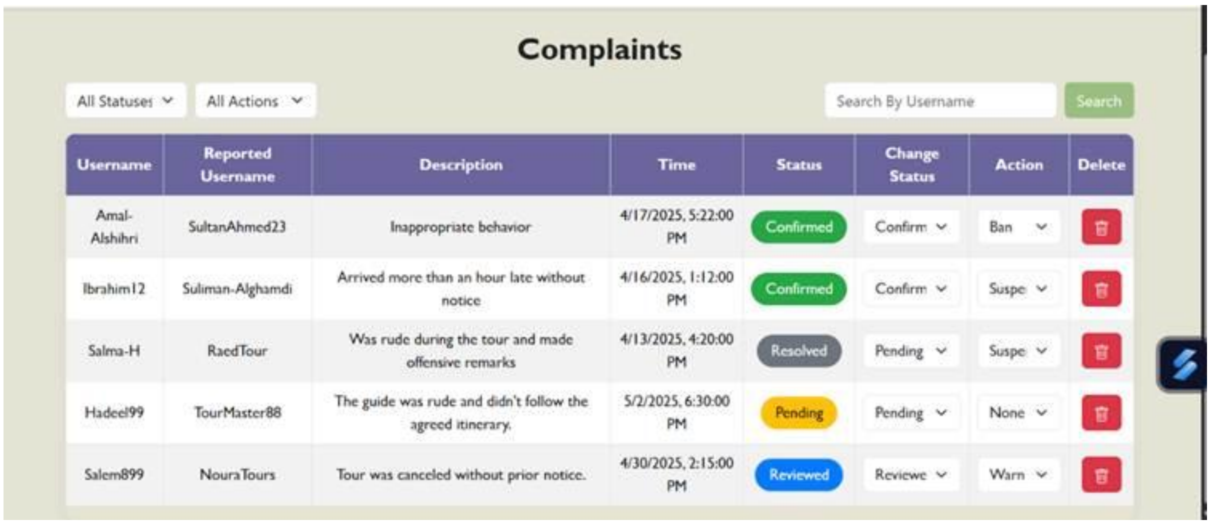
Username	Description	Place	Time	Attachments	Action
Event-Masters@12	Reported behavior	Not Determined	4/17/2025, 5:22:00 PM	View	Select Action
Event-Masters@182	Reported behavior	Not Determined	4/17/2025, 5:22:00 PM	View	Select Action

Complaints Management Page

Lets the admin track, review, and act on user complaints.

- Filters by complaint status and action.
- Search by username.
- Status update and action dropdowns per complaint.
- Delete button for removal.

Tag colors (green = confirmed, yellow = pending) enhance complaint tracking.

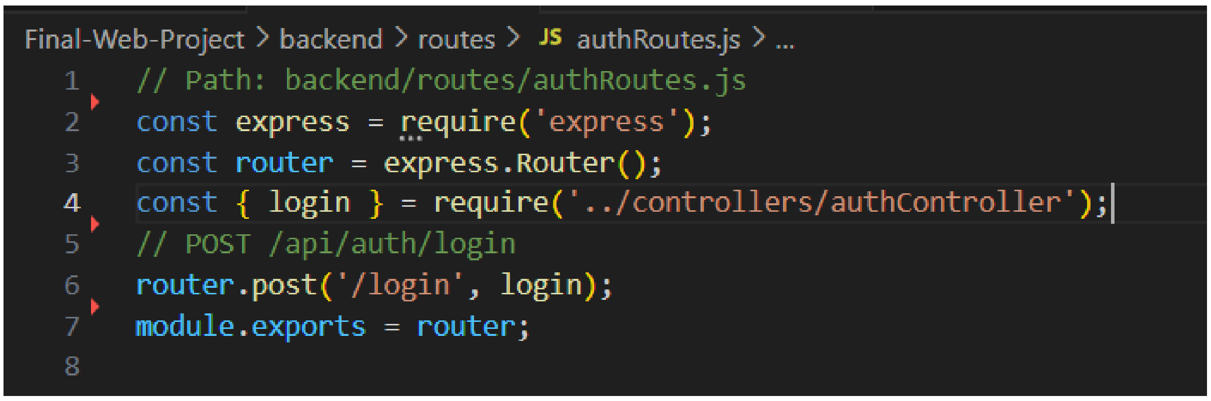


Complaints

Username	Reported Username	Description	Time	Status	Change Status	Action	Delete
Amal-Alshihri	SultanAhmed23	Inappropriate behavior	4/17/2025, 5:22:00 PM	Confirmed	Confirm	Ban	🗑
Ibrahim12	Suliman-Alghamdi	Arrived more than an hour late without notice	4/16/2025, 1:12:00 PM	Confirmed	Confirm	Suspe	🗑
Salma-H	RaedTour	Was rude during the tour and made offensive remarks	4/13/2025, 4:20:00 PM	Resolved	Pending	Suspe	🗑
Hadeel99	TourMaster88	The guide was rude and didn't follow the agreed itinerary.	5/2/2025, 6:30:00 PM	Pending	Pending	None	🗑
Salem899	NouraTours	Tour was canceled without prior notice.	4/30/2025, 2:15:00 PM	Reviewed	Reviewe	Warn	🗑

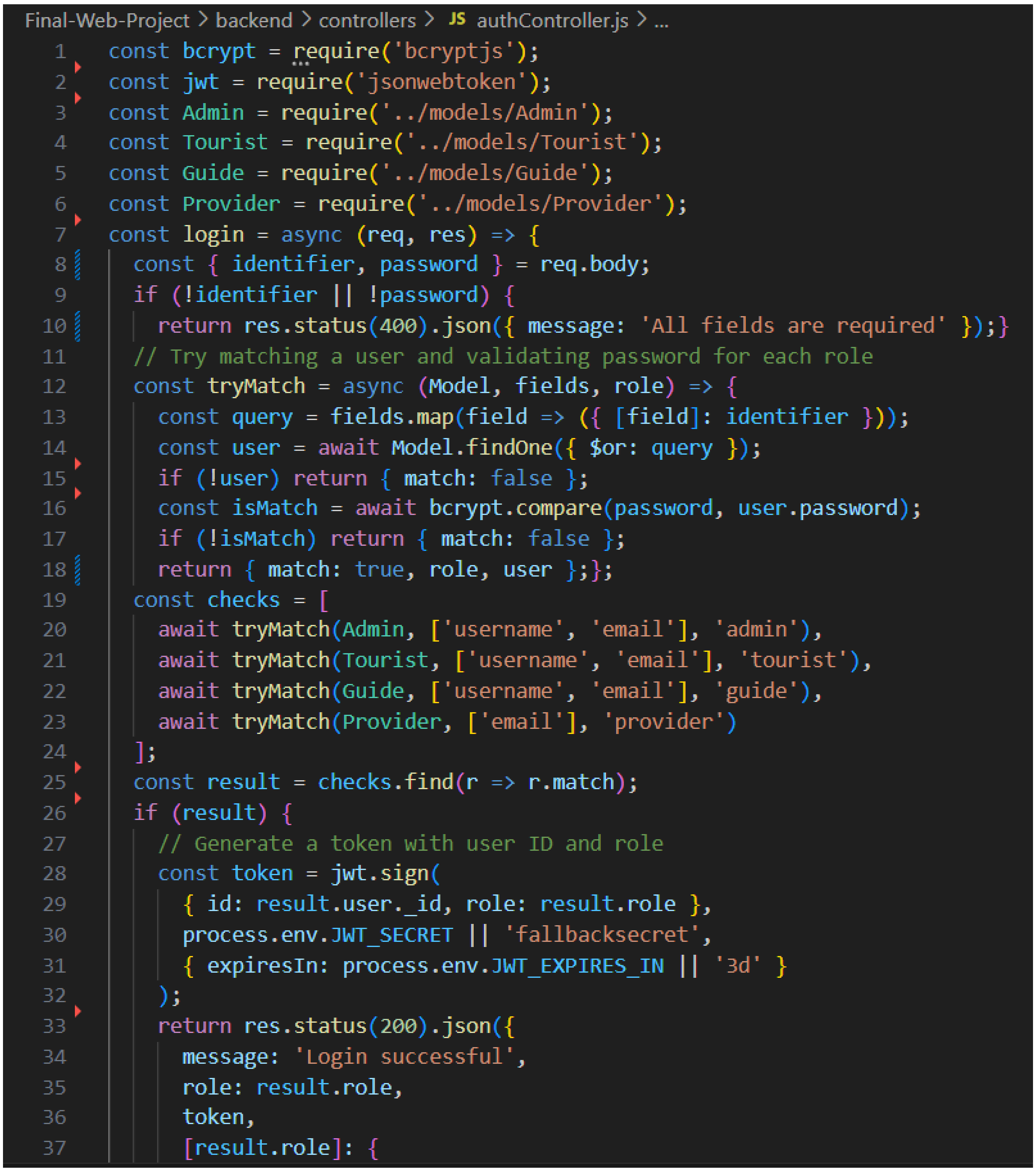
d. Source Code Highlights:

1.



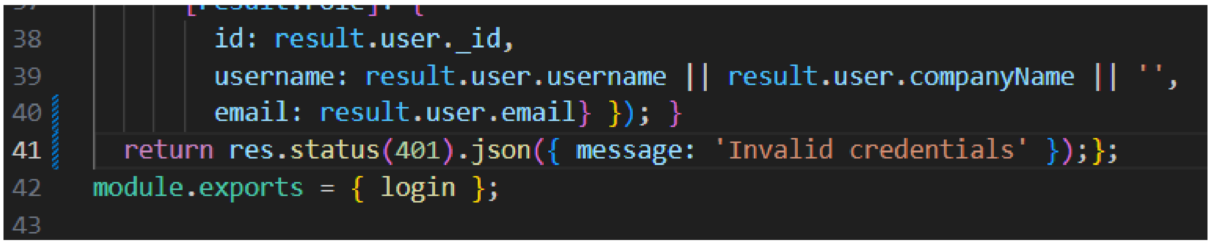
Final-Web-Project > backend > routes > JS authRoutes.js > ...

```javascript
// Path: backend/routes/authRoutes.js
const express = require('express');
const router = express.Router();
const { login } = require('../controllers/authController');
// POST /api/auth/login
router.post('/login', login);
module.exports = router;

```



Final-Web-Project > backend > controllers > JS authController.js > ...

```javascript
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const Admin = require('../models/Admin');
const Tourist = require('../models/Tourist');
const Guide = require('../models/Guide');
const Provider = require('../models/Provider');
const login = async (req, res) => {
  const { identifier, password } = req.body;
  if (!identifier || !password) {
    return res.status(400).json({ message: 'All fields are required' });}
  // Try matching a user and validating password for each role
  const tryMatch = async (Model, fields, role) => {
    const query = fields.map(field => ({ [field]: identifier }));
    const user = await Model.findOne({ $or: query });
    if (!user) return { match: false };
    const isMatch = await bcrypt.compare(password, user.password);
    if (!isMatch) return { match: false };
    return { match: true, role, user };};
  const checks = [
    await tryMatch(Admin, ['username', 'email'], 'admin'),
    await tryMatch(Tourist, ['username', 'email'], 'tourist'),
    await tryMatch(Guide, ['username', 'email'], 'guide'),
    await tryMatch(Provider, ['email'], 'provider')
  ];
  const result = checks.find(r => r.match);
  if (result) {
    // Generate a token with user ID and role
    const token = jwt.sign(
      { id: result.user._id, role: result.role },
      process.env.JWT_SECRET || 'fallbacksecret',
      { expiresIn: process.env.JWT_EXPIRES_IN || '3d' }
    );
    return res.status(200).json({
      message: 'Login successful',
      role: result.role,
      token,
      [result.role]: {
```



```javascript
        id: result.user._id,
        username: result.user.username || result.user.companyName || '',
        email: result.user.email} }); }
  return res.status(401).json({ message: 'Invalid credentials' });};
module.exports = { login };
```

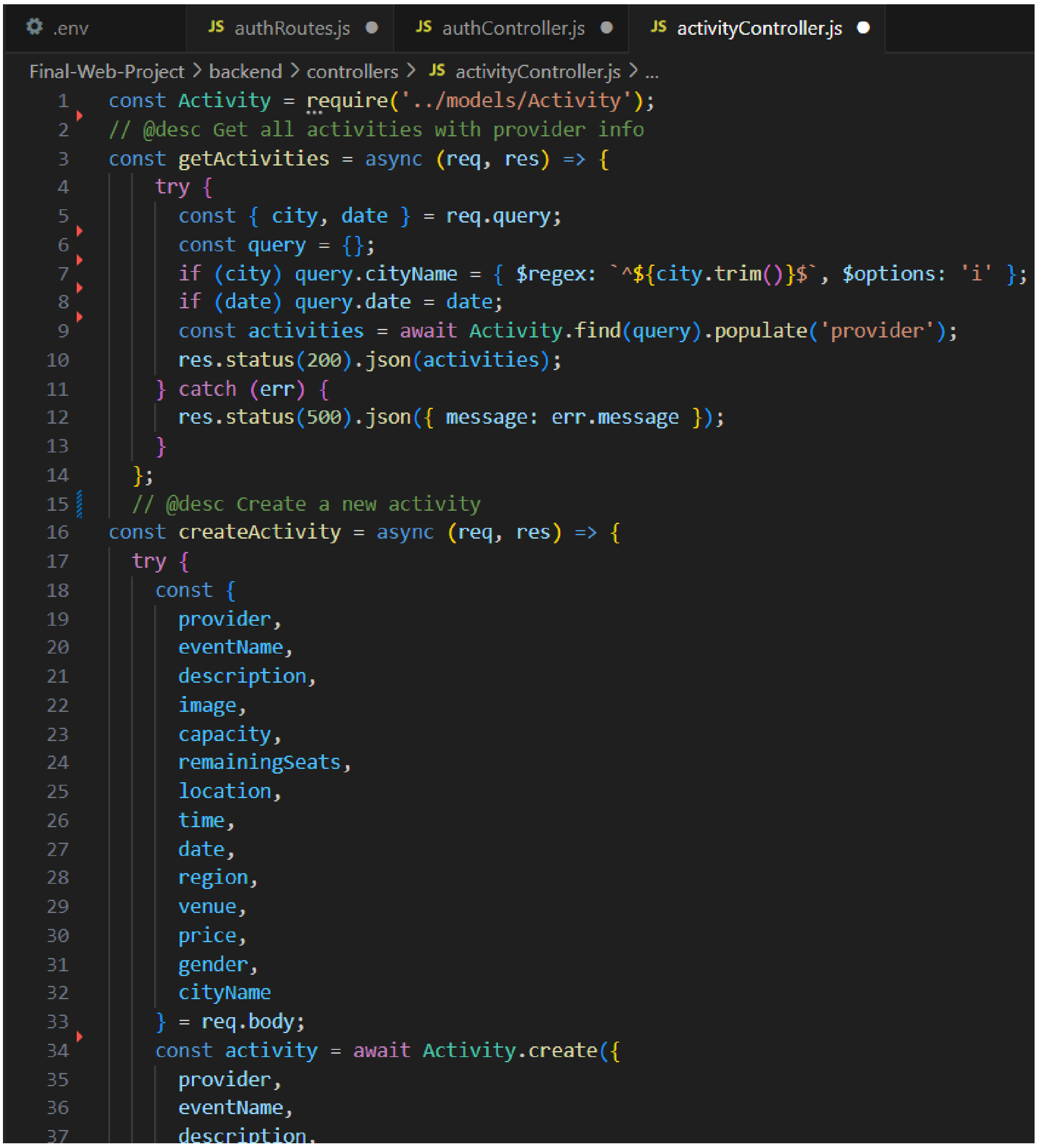
One of the core backend features implemented is the user login system, which is structured using Express.js route handling and a controller function.

In the file backend/routes/authRoutes.js, the /login endpoint is defined to handle POST requests. This route delegates the authentication logic to a login function located in the controller.
The actual login logic is written in backend/controllers/authController.js. It works by:

- Retrieving the user based on the provided email/username.
- Validating the entered password using bcrypt.compare.
- Generating a token upon successful validation.
- Returning the token to the client to establish a logged-in session.

This implementation ensures secure authentication and session management, making it a critical part of the application's backend. And applied to all frontends for each user.

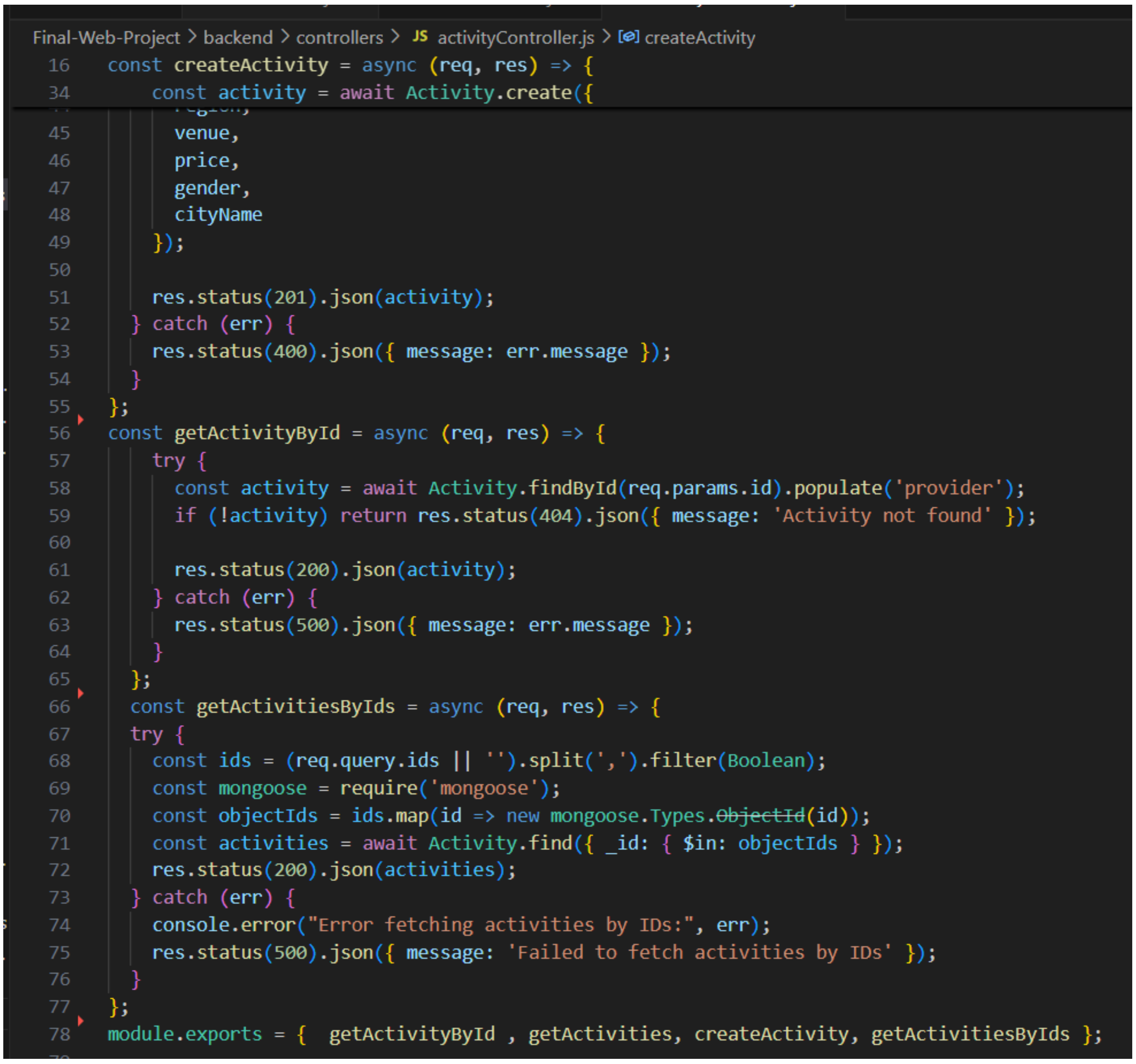
2.



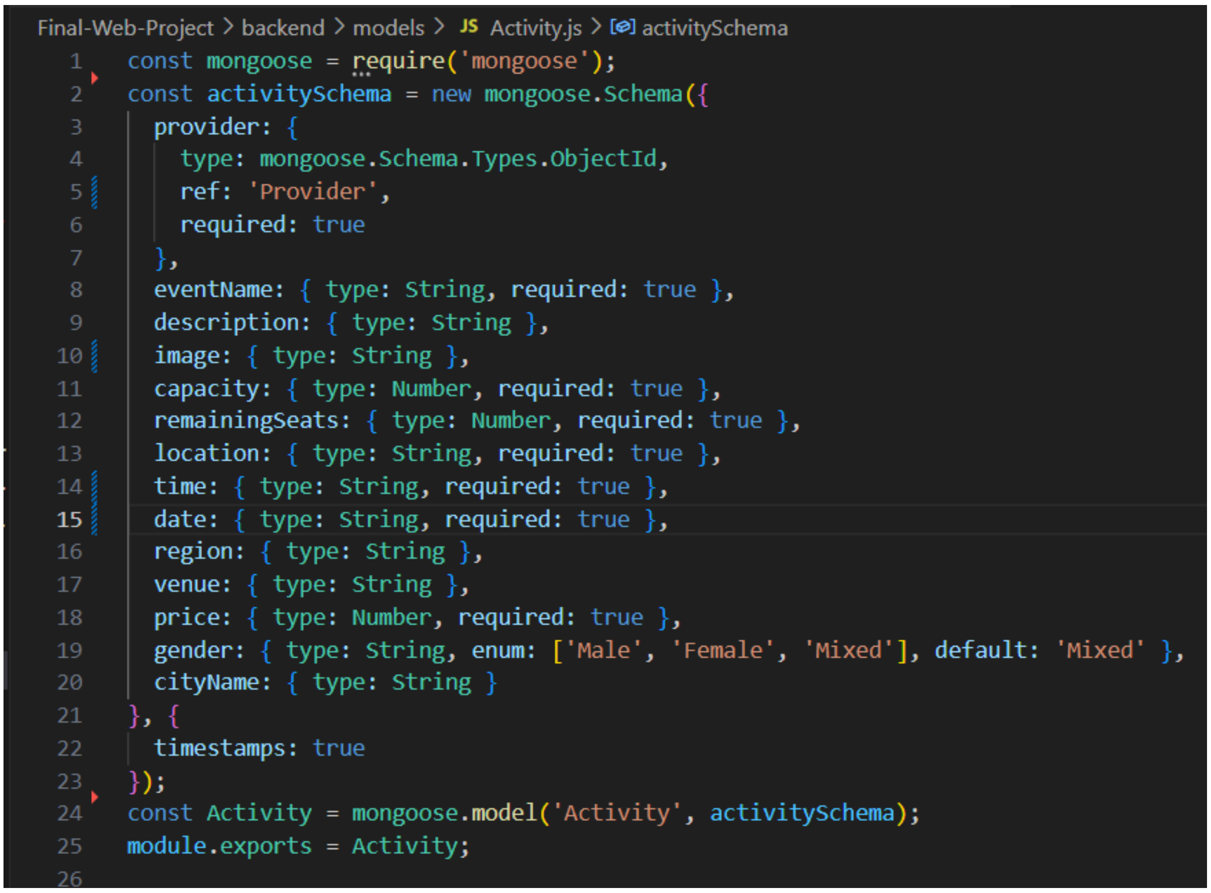
.env   JS authRoutes.js ●   JS authController.js ●   JS activityController.js ●

Final-Web-Project > backend > controllers > JS activityController.js > ...

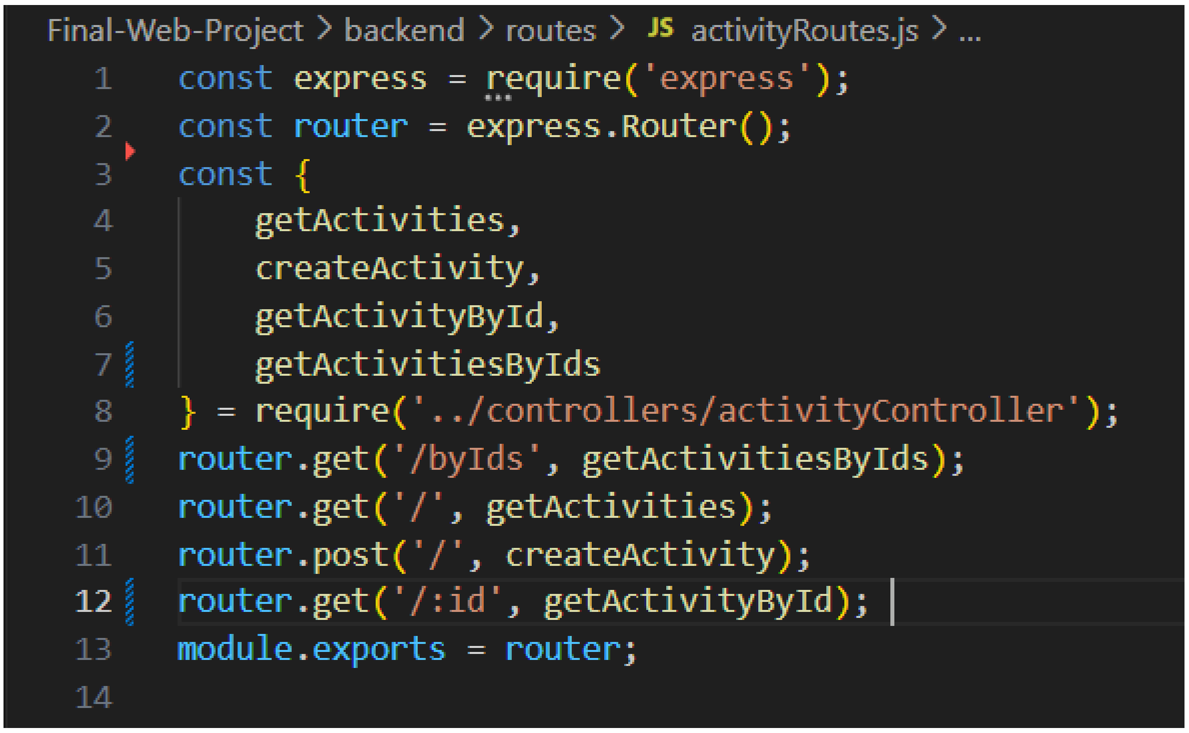
```javascript
const Activity = require('../models/Activity');
// @desc Get all activities with provider info
const getActivities = async (req, res) => {
    try {
      const { city, date } = req.query;
      const query = {};
      if (city) query.cityName = { $regex: `^${city.trim()}$`, $options: 'i' };
      if (date) query.date = date;
      const activities = await Activity.find(query).populate('provider');
      res.status(200).json(activities);
    } catch (err) {
      res.status(500).json({ message: err.message });
    }
  };
  // @desc Create a new activity
const createActivity = async (req, res) => {
  try {
    const {
      provider,
      eventName,
      description,
      image,
      capacity,
      remainingSeats,
      location,
      time,
      date,
      region,
      venue,
      price,
      gender,
      cityName
    } = req.body;
    const activity = await Activity.create({
      provider,
      eventName,
      description,
```


Final-Web-Project > backend > controllers > JS activityController.js > [@] createActivity

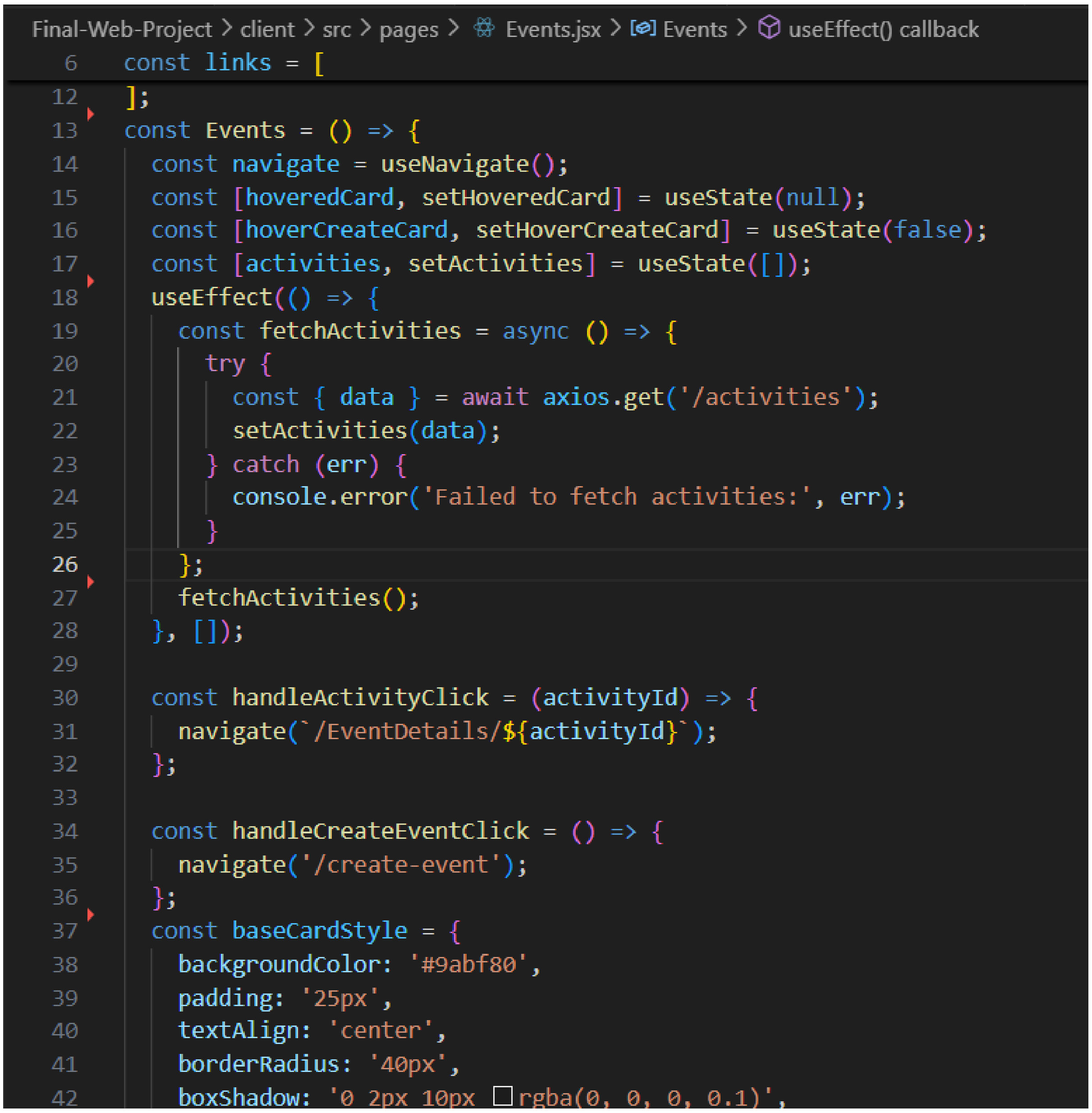
```
const createActivity = async (req, res) => {
    const activity = await Activity.create({
      venue,
      price,
      gender,
      cityName
    });

    res.status(201).json(activity);
  } catch (err) {
    res.status(400).json({ message: err.message });
  }
};
const getActivityById = async (req, res) => {
    try {
      const activity = await Activity.findById(req.params.id).populate('provider');
      if (!activity) return res.status(404).json({ message: 'Activity not found' });

      res.status(200).json(activity);
    } catch (err) {
      res.status(500).json({ message: err.message });
    }
  };
  const getActivitiesByIds = async (req, res) => {
  try {
    const ids = (req.query.ids || '').split(',').filter(Boolean);
    const mongoose = require('mongoose');
    const objectIds = ids.map(id => new mongoose.Types.~~ObjectId~~(id));
    const activities = await Activity.find({ _id: { $in: objectIds } });
    res.status(200).json(activities);
  } catch (err) {
    console.error("Error fetching activities by IDs:", err);
    res.status(500).json({ message: 'Failed to fetch activities by IDs' });
  }
};
module.exports = {  getActivityById , getActivities, createActivity, getActivitiesByIds };
```


Final-Web-Project > backend > models > JS Activity.js > activitySchema

```
const mongoose = require('mongoose');
const activitySchema = new mongoose.Schema({
  provider: {
    type: mongoose.Schema.Types.ObjectId,
    ref: 'Provider',
    required: true
  },
  eventName: { type: String, required: true },
  description: { type: String },
  image: { type: String },
  capacity: { type: Number, required: true },
  remainingSeats: { type: Number, required: true },
  location: { type: String, required: true },
  time: { type: String, required: true },
  date: { type: String, required: true },
  region: { type: String },
  venue: { type: String },
  price: { type: Number, required: true },
  gender: { type: String, enum: ['Male', 'Female', 'Mixed'], default: 'Mixed' },
  cityName: { type: String }
}, {
  timestamps: true
});
const Activity = mongoose.model('Activity', activitySchema);
module.exports = Activity;
```


Final-Web-Project > backend > routes > JS activityRoutes.js > ...

```
const express = require('express');
const router = express.Router();
const {
    getActivities,
    createActivity,
    getActivityById,
    getActivitiesByIds
} = require('../controllers/activityController');
router.get('/byIds', getActivitiesByIds);
router.get('/', getActivities);
router.post('/', createActivity);
router.get('/:id', getActivityById);
module.exports = router;
```



Final-Web-Project > client > src > pages > Events.jsx > Events > useEffect() callback

```
const links = [
];
const Events = () => {
  const navigate = useNavigate();
  const [hoveredCard, setHoveredCard] = useState(null);
  const [hoverCreateCard, setHoverCreateCard] = useState(false);
  const [activities, setActivities] = useState([]);
  useEffect(() => {
    const fetchActivities = async () => {
      try {
        const { data } = await axios.get('/activities');
        setActivities(data);
      } catch (err) {
        console.error('Failed to fetch activities:', err);
      }
    };
    fetchActivities();
  }, []);

  const handleActivityClick = (activityId) => {
    navigate(`/EventDetails/${activityId}`);
  };

  const handleCreateEventClick = () => {
    navigate('/create-event');
  };
  const baseCardStyle = {
    backgroundColor: '#9abf80',
    padding: '25px',
    textAlign: 'center',
    borderRadius: '40px',
    boxShadow: '0 2px 10px rgba(0, 0, 0, 0.1)',
```

One of the core backend features implemented is the activity management system, which allows providers to create, retrieve, and manage events (activities) in the platform.

In the file backend/routes/activityRoutes.js, multiple endpoints are defined:

- A POST / route for creating a new activity.
- A GET / route for retrieving all activities, with optional filters.
- A GET /:id route for retrieving a specific activity by its ID.

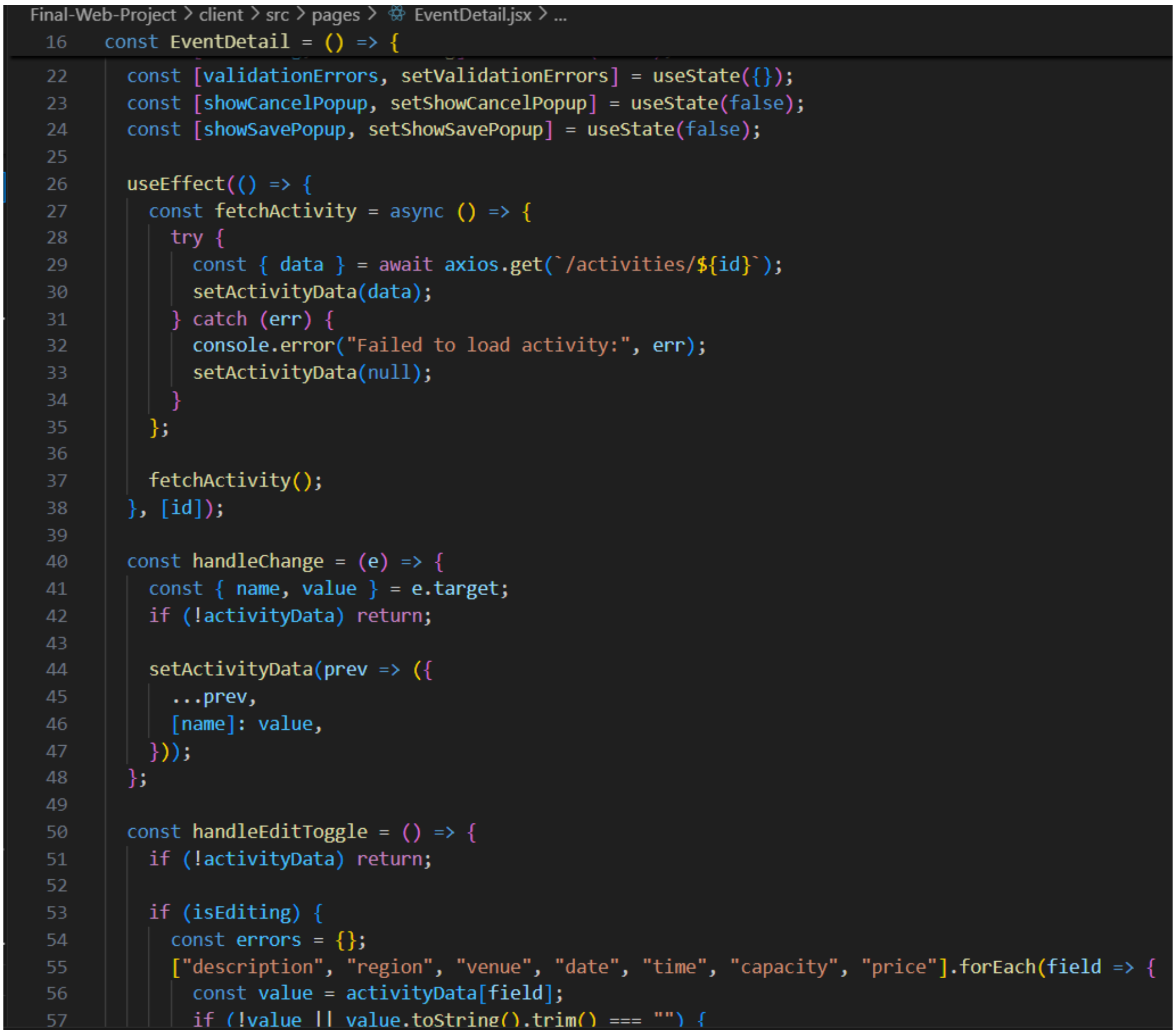
These routes delegate logic to their respective functions in the activity controller.

The main logic for creating and fetching activities is written in backend/controllers/activityController.js. It works by:

- Receiving activity data from the frontend through the request body.
- Validating and saving the data to MongoDB using Mongoose.
- Optionally filtering activities based on city or date via query parameters.
- Populating the related provider field when retrieving activity data.

On the frontend, in the file client/src/pages/Events.jsx, a useEffect hook runs on page load to fetch the list of available activities from the backend using axios.get('/activities'). The retrieved data is stored in component state and displayed in the UI as interactive cards.
When a user clicks on an activity card, they are navigated to its detail page using navigate('/EventDetails/:id').

3.



```
Final-Web-Project > client > src > pages > EventDetail.jsx > ...
const EventDetail = () => {
    const [validationErrors, setValidationErrors] = useState({});
    const [showCancelPopup, setShowCancelPopup] = useState(false);
    const [showSavePopup, setShowSavePopup] = useState(false);

    useEffect(() => {
      const fetchActivity = async () => {
        try {
          const { data } = await axios.get(`/activities/${id}`);
          setActivityData(data);
        } catch (err) {
          console.error("Failed to load activity:", err);
          setActivityData(null);
        }
      };

      fetchActivity();
    }, [id]);

    const handleChange = (e) => {
      const { name, value } = e.target;
      if (!activityData) return;

      setActivityData(prev => ({
        ...prev,
        [name]: value,
      }));
    };

    const handleEditToggle = () => {
      if (!activityData) return;

      if (isEditing) {
        const errors = {};
        ["description", "region", "venue", "date", "time", "capacity", "price"].forEach(field => {
          const value = activityData[field];
          if (!value || value.toString().trim() === "") {
```

One of the key features available to the activity provider is to view full details of a selected activity

When the component mounts, it uses the useEffect hook to trigger an API integration:

- This API call fetches a specific activity from the backend using its ID.
- The backend controller executes a MongoDB database query (Activity.findById(id)) to retrieve that activity and return it as a response.

The retrieved data is stored in local state (activityData) and rendered dynamically on the page.

The component also features real-time form handling logic

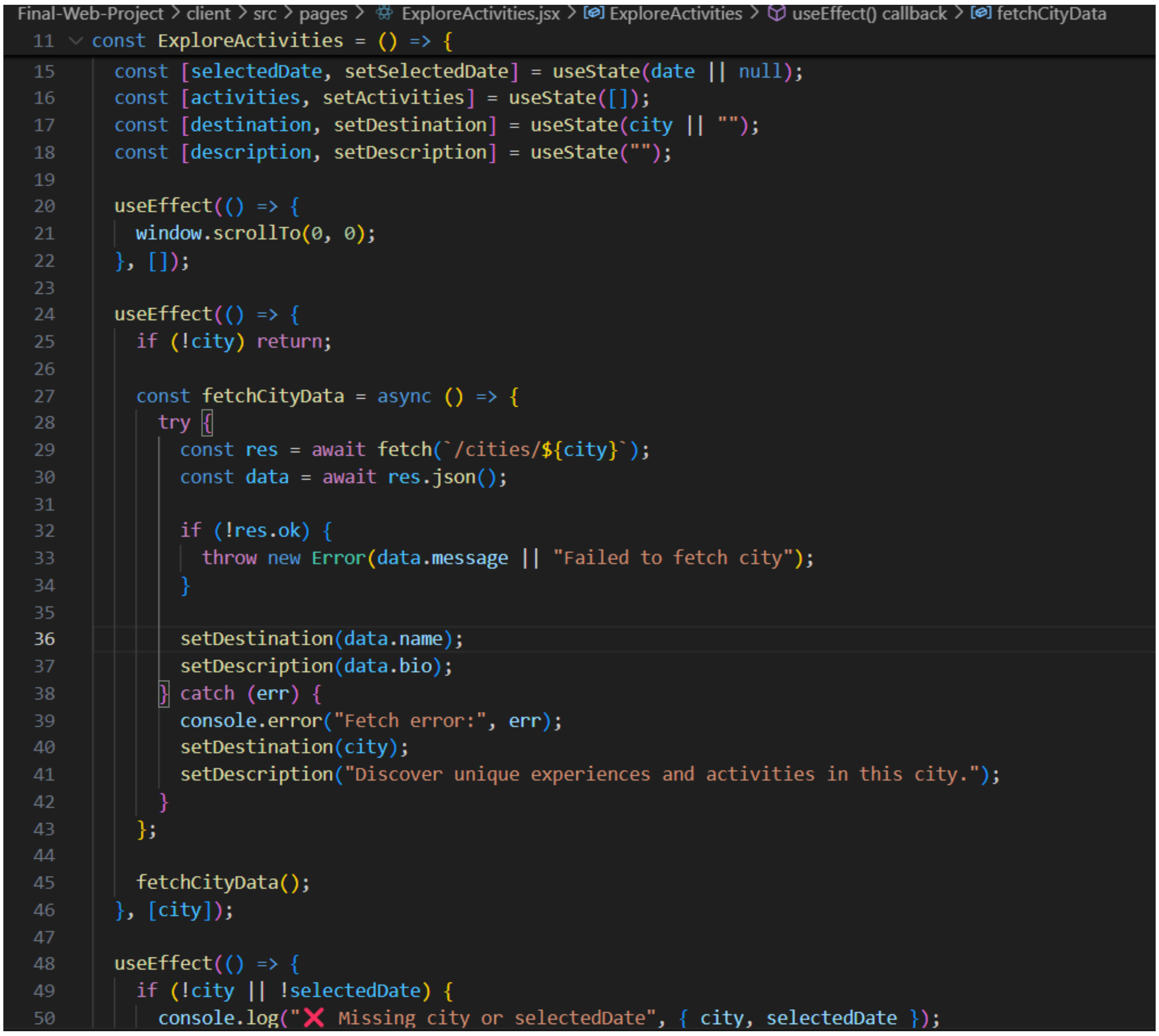
This simple state update algorithm enables users to edit form values field-by-field.

Before saving, handleEditToggle() performs input validation using a basic field-checking algorithm

This loop ensures that no required fields are left empty and that the data structure complies with backend expectations.

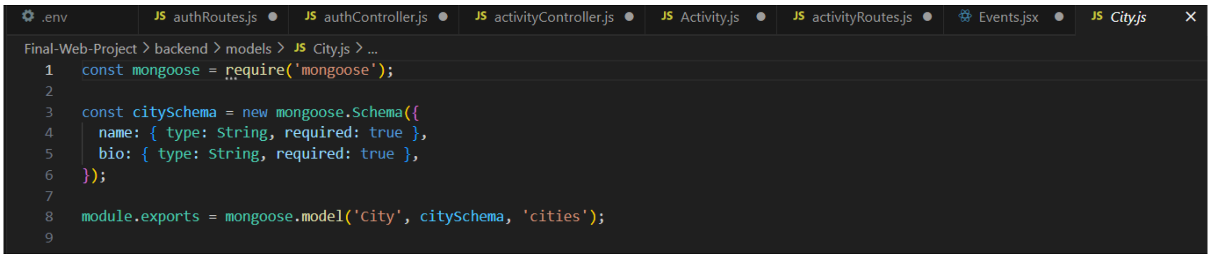
> This feature demonstrates frontend-backend coordination through RESTful API calls, use of form-handling logic, and data validation algorithms, making it a key interactive component in the system.

4.

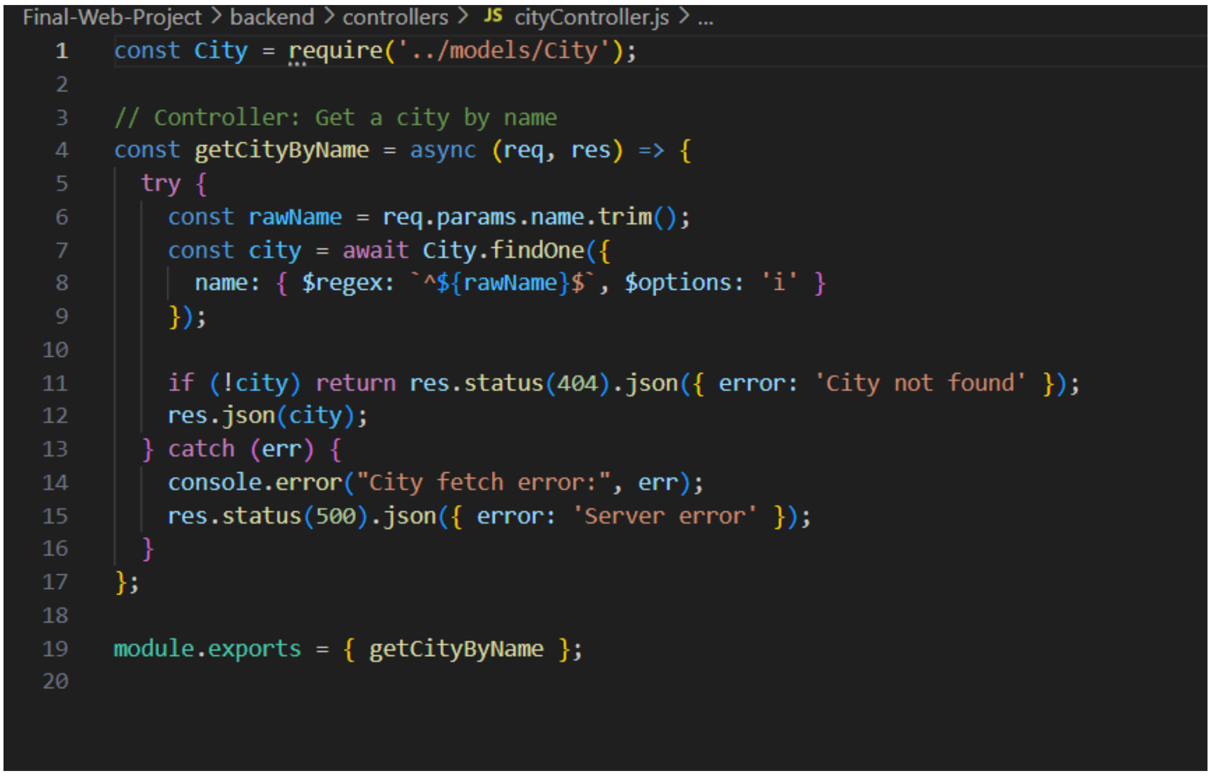


Final-Web-Project > client > src > pages > ExploreActivities.jsx > ExploreActivities > useEffect() callback > fetchCityData

```jsx
const ExploreActivities = () => {
  const [selectedDate, setSelectedDate] = useState(date || null);
  const [activities, setActivities] = useState([]);
  const [destination, setDestination] = useState(city || "");
  const [description, setDescription] = useState("");

  useEffect(() => {
    window.scrollTo(0, 0);
  }, []);

  useEffect(() => {
    if (!city) return;

    const fetchCityData = async () => {
      try {
        const res = await fetch(`/cities/${city}`);
        const data = await res.json();

        if (!res.ok) {
          throw new Error(data.message || "Failed to fetch city");
        }

        setDestination(data.name);
        setDescription(data.bio);
      } catch (err) {
        console.error("Fetch error:", err);
        setDestination(city);
        setDescription("Discover unique experiences and activities in this city.");
      }
    };

    fetchCityData();
  }, [city]);

  useEffect(() => {
    if (!city || !selectedDate) {
      console.log("❌ Missing city or selectedDate", { city, selectedDate });
```



```javascript
const mongoose = require('mongoose');

const citySchema = new mongoose.Schema({
  name: { type: String, required: true },
  bio: { type: String, required: true },
});

module.exports = mongoose.model('City', citySchema, 'cities');
```


Final-Web-Project > backend > controllers > cityController.js > ...

```javascript
const City = require('../models/City');

// Controller: Get a city by name
const getCityByName = async (req, res) => {
  try {
    const rawName = req.params.name.trim();
    const city = await City.findOne({
      name: { $regex: `^${rawName}$`, $options: 'i' }
    });

    if (!city) return res.status(404).json({ error: 'City not found' });
    res.json(city);
  } catch (err) {
    console.error("City fetch error:", err);
    res.status(500).json({ error: 'Server error' });
  }
};

module.exports = { getCityByName };
```

A key feature for the tourist in the ExploreActivities.jsx is the dynamic retrieval of city information. This functionality enhances user experience by displaying a relevant name and description for the selected city before listing any activities.

When the component detects a selected city, it triggers the fetchCityData() function via the useEffect hook. This function performs an API integration using the native fetch API:

This request is sent to the backend to retrieve metadata about the city (name and bio). If the request fails, the component falls back to a default description, ensuring graceful error handling.

The retrieved data is stored in React state using setDestination() and setDescription() to dynamically populate the hero banner area of the UI.

This functionality shows the use of:

- API integration for content enrichment
- Asynchronous logic and error handling algorithms
- Real-time UI updates based on backend responses

It ensures that each tourist browsing a location gets a personalized and informative interface, tailored to the city they are exploring.

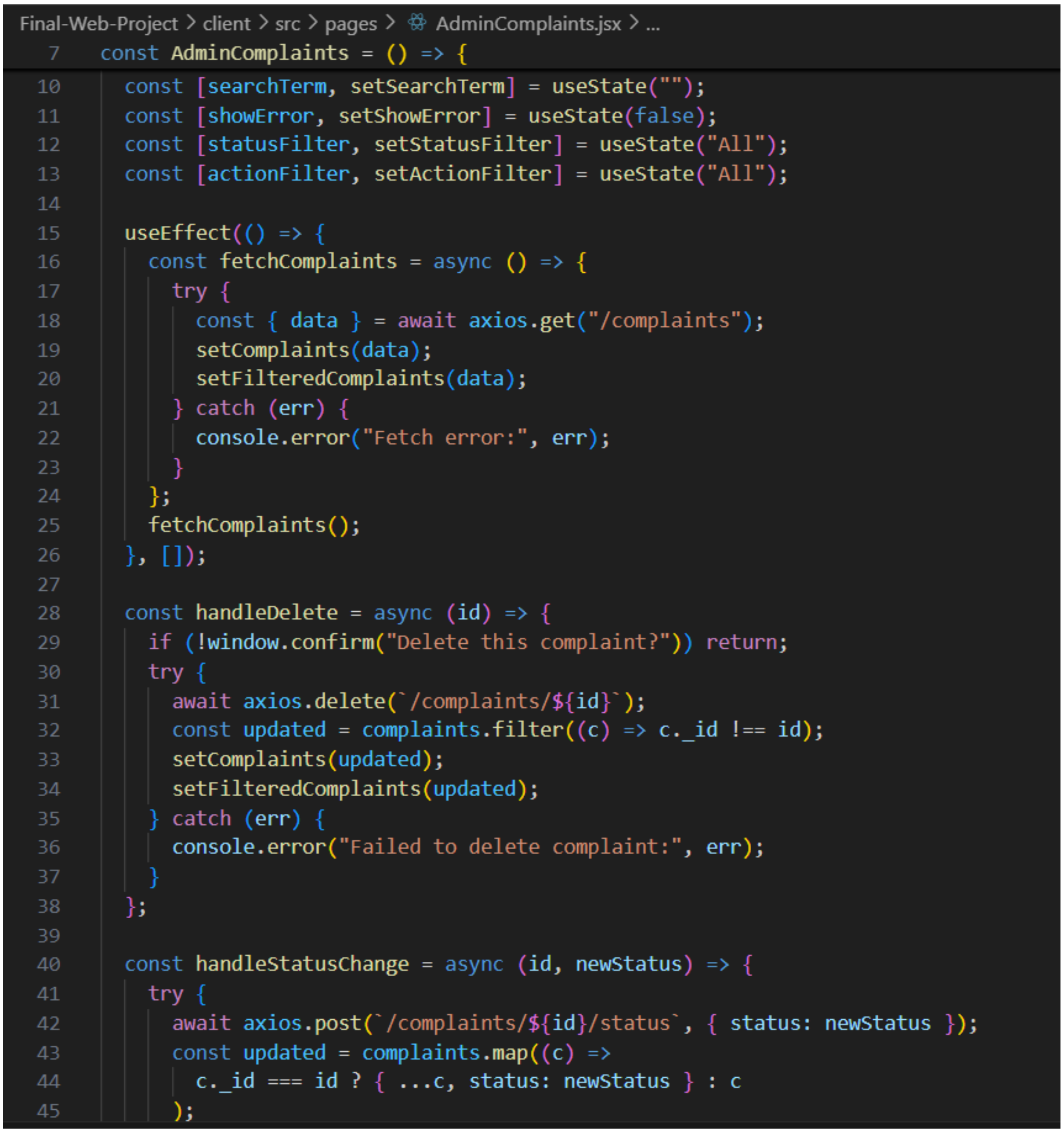
For the backend: this query uses a case-insensitive regular expression to match the city name exactly, allowing for flexible user input.

If the city is found, it is returned as a JSON response. If not, the server responds with a 404 error. The function includes robust error handling, ensuring that any internal issues are logged and returned as a 500 response to the frontend.

This functionality demonstrates:

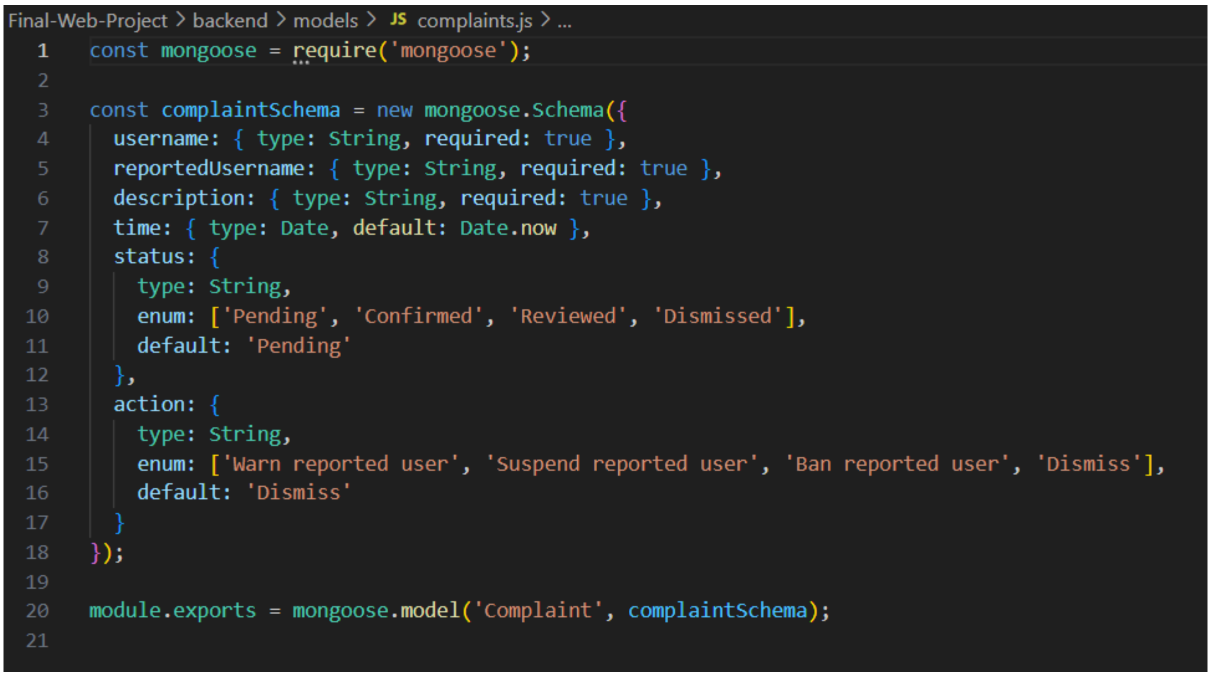
- Use of database queries with regex filters.
- A simple but effective string-matching algorithm.
- Clean API integration supporting dynamic UI behavior for tourists

5.



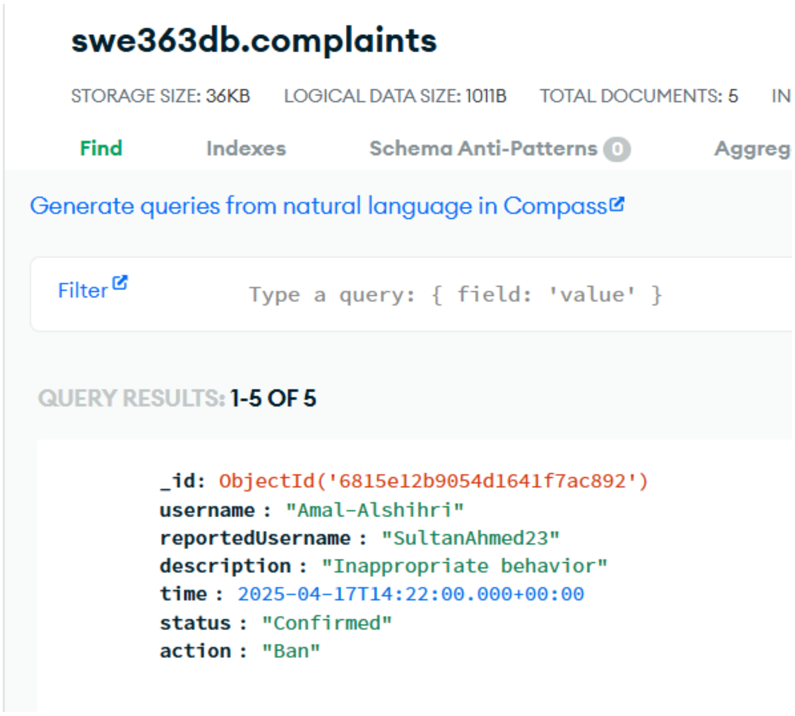
Final-Web-Project > client > src > pages > AdminComplaints.jsx > ...

```
const AdminComplaints = () => {
  const [searchTerm, setSearchTerm] = useState("");
  const [showError, setShowError] = useState(false);
  const [statusFilter, setStatusFilter] = useState("All");
  const [actionFilter, setActionFilter] = useState("All");

  useEffect(() => {
    const fetchComplaints = async () => {
      try {
        const { data } = await axios.get("/complaints");
        setComplaints(data);
        setFilteredComplaints(data);
      } catch (err) {
        console.error("Fetch error:", err);
      }
    };
    fetchComplaints();
  }, []);

  const handleDelete = async (id) => {
    if (!window.confirm("Delete this complaint?")) return;
    try {
      await axios.delete(`/complaints/${id}`);
      const updated = complaints.filter((c) => c._id !== id);
      setComplaints(updated);
      setFilteredComplaints(updated);
    } catch (err) {
      console.error("Failed to delete complaint:", err);
    }
  };

  const handleStatusChange = async (id, newStatus) => {
    try {
      await axios.post(`/complaints/${id}/status`, { status: newStatus });
      const updated = complaints.map((c) =>
        c._id === id ? { ...c, status: newStatus } : c
      );
```



Final-Web-Project > backend > models > JS complaints.js > ...

```javascript
const mongoose = require('mongoose');

const complaintSchema = new mongoose.Schema({
  username: { type: String, required: true },
  reportedUsername: { type: String, required: true },
  description: { type: String, required: true },
  time: { type: Date, default: Date.now },
  status: {
    type: String,
    enum: ['Pending', 'Confirmed', 'Reviewed', 'Dismissed'],
    default: 'Pending'
  },
  action: {
    type: String,
    enum: ['Warn reported user', 'Suspend reported user', 'Ban reported user', 'Dismiss'],
    default: 'Dismiss'
  }
});

module.exports = mongoose.model('Complaint', complaintSchema);
```



swe363db.complaints

STORAGE SIZE: 36KB   LOGICAL DATA SIZE: 1011B   TOTAL DOCUMENTS: 5   IN

Find
Indexes
Schema Anti-Patterns 0
Aggreg
Generate queries from natural language in Compass
Filter
Type a query: { field: 'value' }

QUERY RESULTS: 1-5 OF 5

```
_id: ObjectId('6815e12b9054d1641f7ac892')
username : "Amal-Alshihri"
reportedUsername : "SultanAhmed23"
description : "Inappropriate behavior"
time : 2025-04-17T14:22:00.000+00:00
status : "Confirmed"
action : "Ban"
```

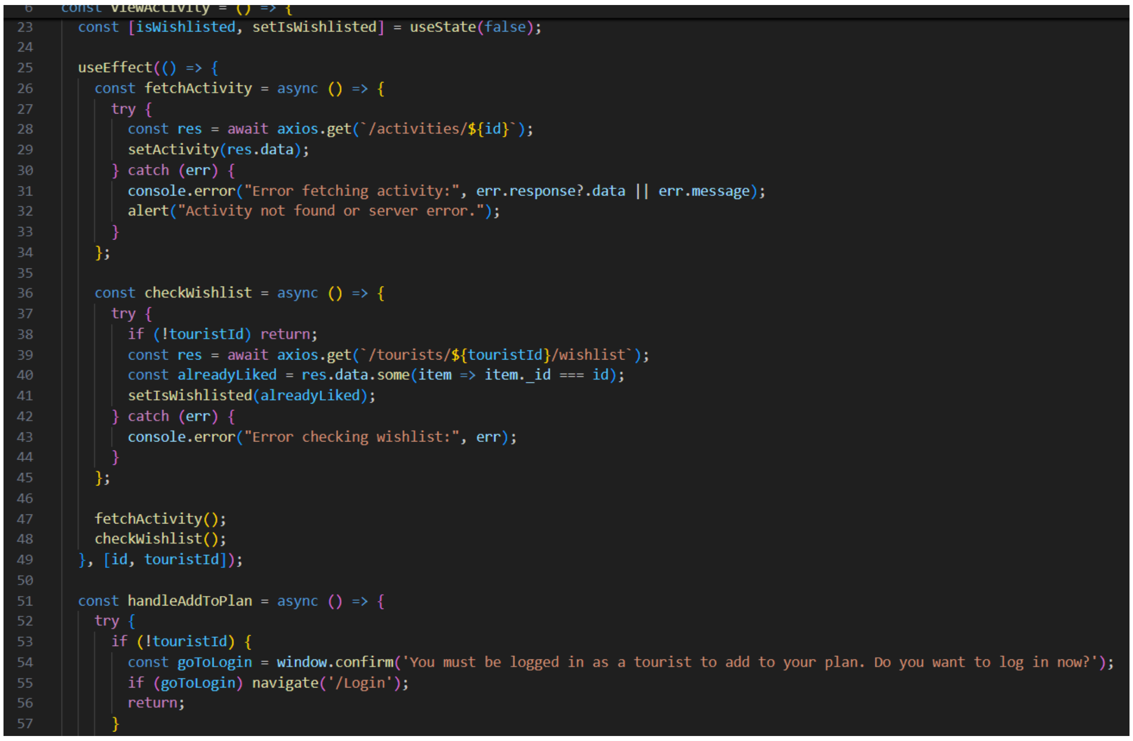
The complaint management that allows admins to review and act on user-submitted reports. Complaints are stored in MongoDB using a structured schema that includes fields such as username, reportedUsername, description, status, and action.

In the AdminComplaints.jsx page, complaints are fetched using an API call to /complaints and displayed in a list. Admins can change a complaint’s status (e.g., from "Pending" to "Confirmed") using a POST request, or delete complaints entirely via a DELETE request. These actions update both the database and the frontend interface.

The system uses Axios for API integration and Mongoose for database interaction. Data validation is enforced through schema enums, ensuring consistency for fields.

This feature enables effective admin moderation, combining database queries, status-handling logic, and real-time UI updates.

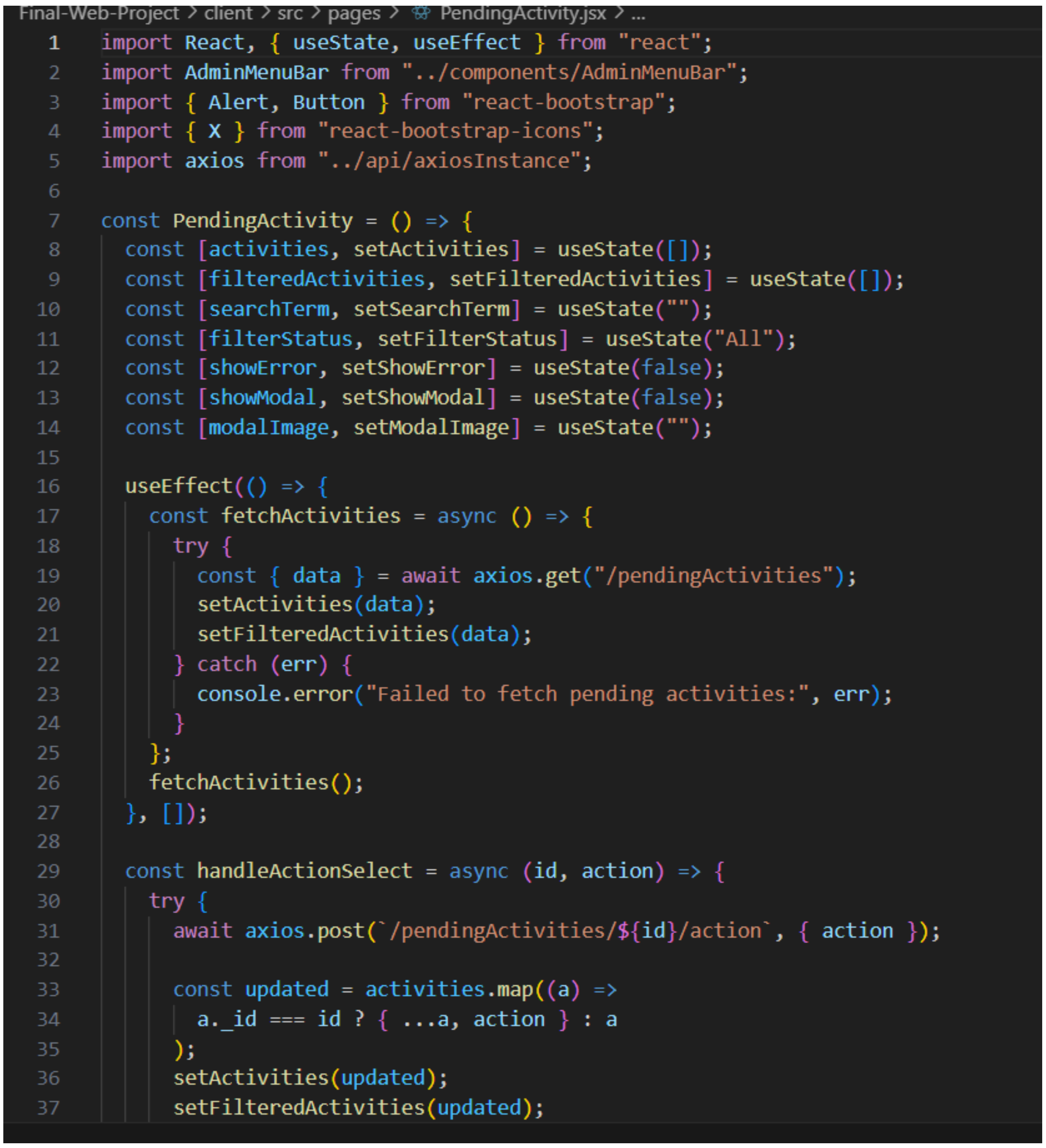
6.



```
const ViewActivity = () => {
  const [isWishlisted, setIsWishlisted] = useState(false);

  useEffect(() => {
    const fetchActivity = async () => {
      try {
        const res = await axios.get(`/activities/${id}`);
        setActivity(res.data);
      } catch (err) {
        console.error("Error fetching activity:", err.response?.data || err.message);
        alert("Activity not found or server error.");
      }
    };

    const checkWishlist = async () => {
      try {
        if (!touristId) return;
        const res = await axios.get(`/tourists/${touristId}/wishlist`);
        const alreadyLiked = res.data.some(item => item._id === id);
        setIsWishlisted(alreadyLiked);
      } catch (err) {
        console.error("Error checking wishlist:", err);
      }
    };

    fetchActivity();
    checkWishlist();
  }, [id, touristId]);

  const handleAddToPlan = async () => {
    try {
      if (!touristId) {
        const goToLogin = window.confirm('You must be logged in as a tourist to add to your plan. Do you want to log in now?');
        if (goToLogin) navigate('/Login');
        return;
      }
```

The ViewActivity component allows tourists to view activity details and manage their wishlist. When the page loads, it fetches the activity using /activities/:id and checks if it's in the tourist's wishlist via /tourists/:touristId/wishlist. A simple .some() check determines if the activity is already liked.

If a user tries to add an activity without being logged in, a prompt appears, and they are redirected to the login page. This feature combines API calls, array-matching logic, and session checks to enhance the tourist experience.



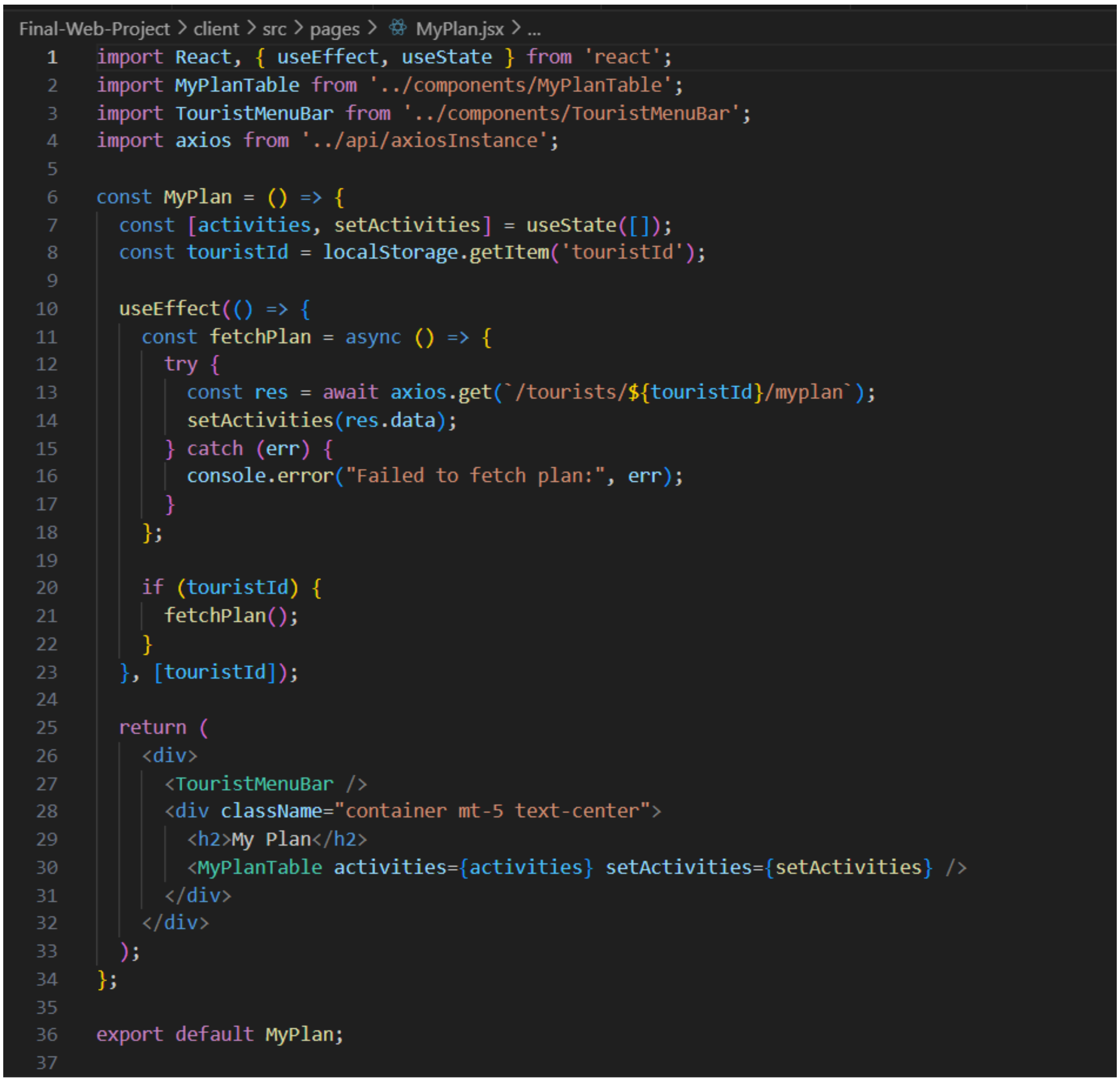
Final-Web-Project > client > src > pages > PendingActivity.jsx > ...

```jsx
import React, { useState, useEffect } from "react";
import AdminMenuBar from "../components/AdminMenuBar";
import { Alert, Button } from "react-bootstrap";
import { X } from "react-bootstrap-icons";
import axios from "../api/axiosInstance";

const PendingActivity = () => {
  const [activities, setActivities] = useState([]);
  const [filteredActivities, setFilteredActivities] = useState([]);
  const [searchTerm, setSearchTerm] = useState("");
  const [filterStatus, setFilterStatus] = useState("All");
  const [showError, setShowError] = useState(false);
  const [showModal, setShowModal] = useState(false);
  const [modalImage, setModalImage] = useState("");

  useEffect(() => {
    const fetchActivities = async () => {
      try {
        const { data } = await axios.get("/pendingActivities");
        setActivities(data);
        setFilteredActivities(data);
      } catch (err) {
        console.error("Failed to fetch pending activities:", err);
      }
    };
    fetchActivities();
  }, []);

  const handleActionSelect = async (id, action) => {
    try {
      await axios.post(`/pendingActivities/${id}/action`, { action });

      const updated = activities.map((a) =>
        a._id === id ? { ...a, action } : a
      );
      setActivities(updated);
      setFilteredActivities(updated);
```

The PendingActivity.jsx page allows admins to review and act on newly submitted activities. On load, it fetches all pending activities using a GET request to /pendingActivities, storing them in state for filtering and display.

Admins can approve or reject activities through a dropdown. When an action is selected, a POST request is sent to /pendingActivities/:id/action with the chosen value. The local state is updated using a .map() algorithm to reflect the change in real time.

This feature highlights the use of API integration, conditional UI updates, and stateful list manipulation to enable smooth and responsive admin moderation.



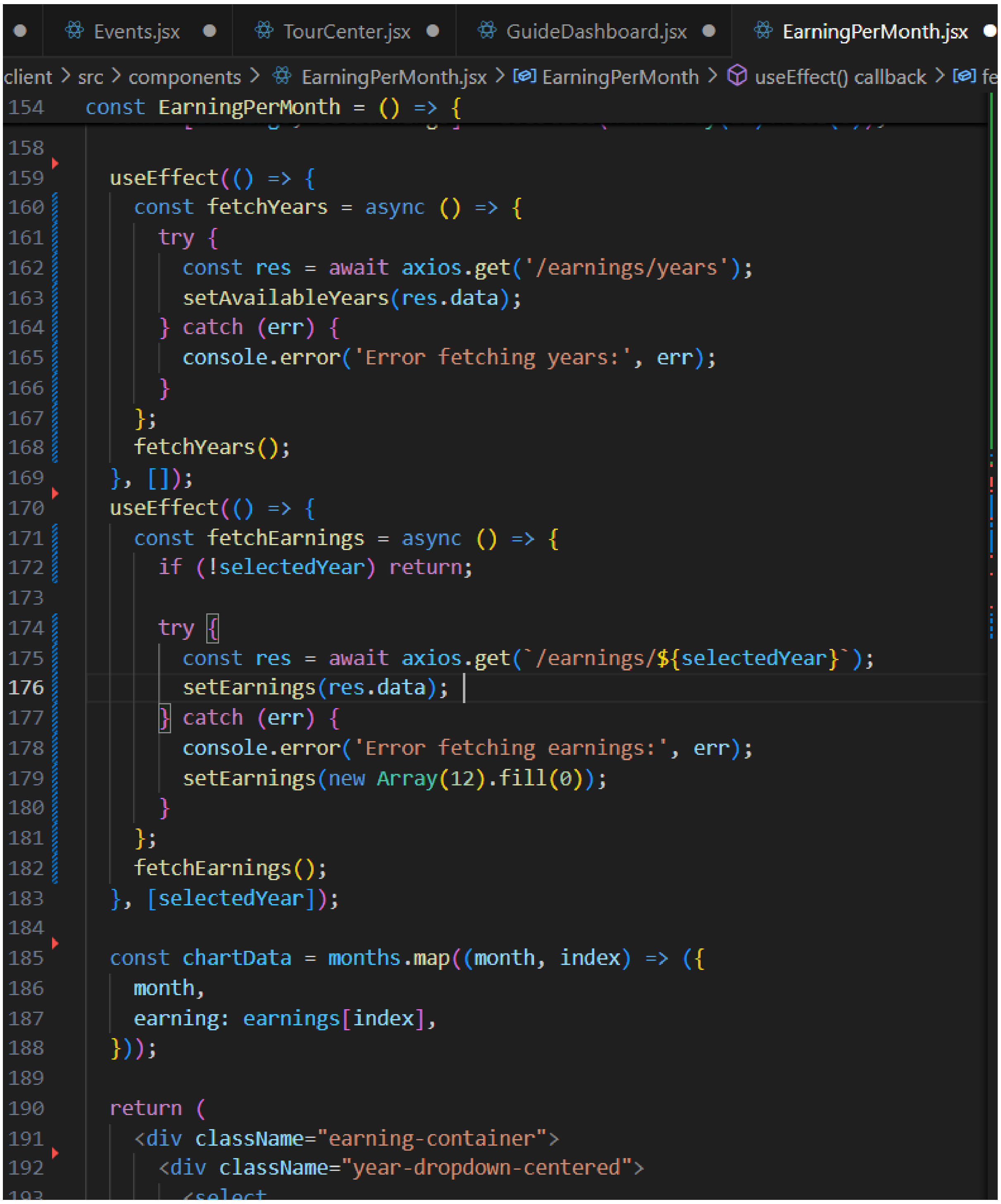
Final-Web-Project > client > src > pages > MyPlan.jsx > ...

```jsx
import React, { useEffect, useState } from 'react';
import MyPlanTable from '../components/MyPlanTable';
import TouristMenuBar from '../components/TouristMenuBar';
import axios from '../api/axiosInstance';

const MyPlan = () => {
  const [activities, setActivities] = useState([]);
  const touristId = localStorage.getItem('touristId');

  useEffect(() => {
    const fetchPlan = async () => {
      try {
        const res = await axios.get(`/tourists/${touristId}/myplan`);
        setActivities(res.data);
      } catch (err) {
        console.error("Failed to fetch plan:", err);
      }
    };

    if (touristId) {
      fetchPlan();
    }
  }, [touristId]);

  return (
    <div>
      <TouristMenuBar />
      <div className="container mt-5 text-center">
        <h2>My Plan</h2>
        <MyPlanTable activities={activities} setActivities={setActivities} />
      </div>
    </div>
  );
};

export default MyPlan;
```

The MyPlan page allows tourists to view a list of activities they have added to their personal plan. When the component loads, it retrieves the tourist's ID from local storage and sends a GET request to /tourists/:touristId/myplan.

The returned activity data is stored in state and rendered in a table using the MyPlanTable component. This feature integrates API requests, local storage retrieval, and dynamic UI updates, enabling personalized trip planning functionality for tourists.

7.



Events.jsx TourCenter.jsx GuideDashboard.jsx EarningPerMonth.jsx

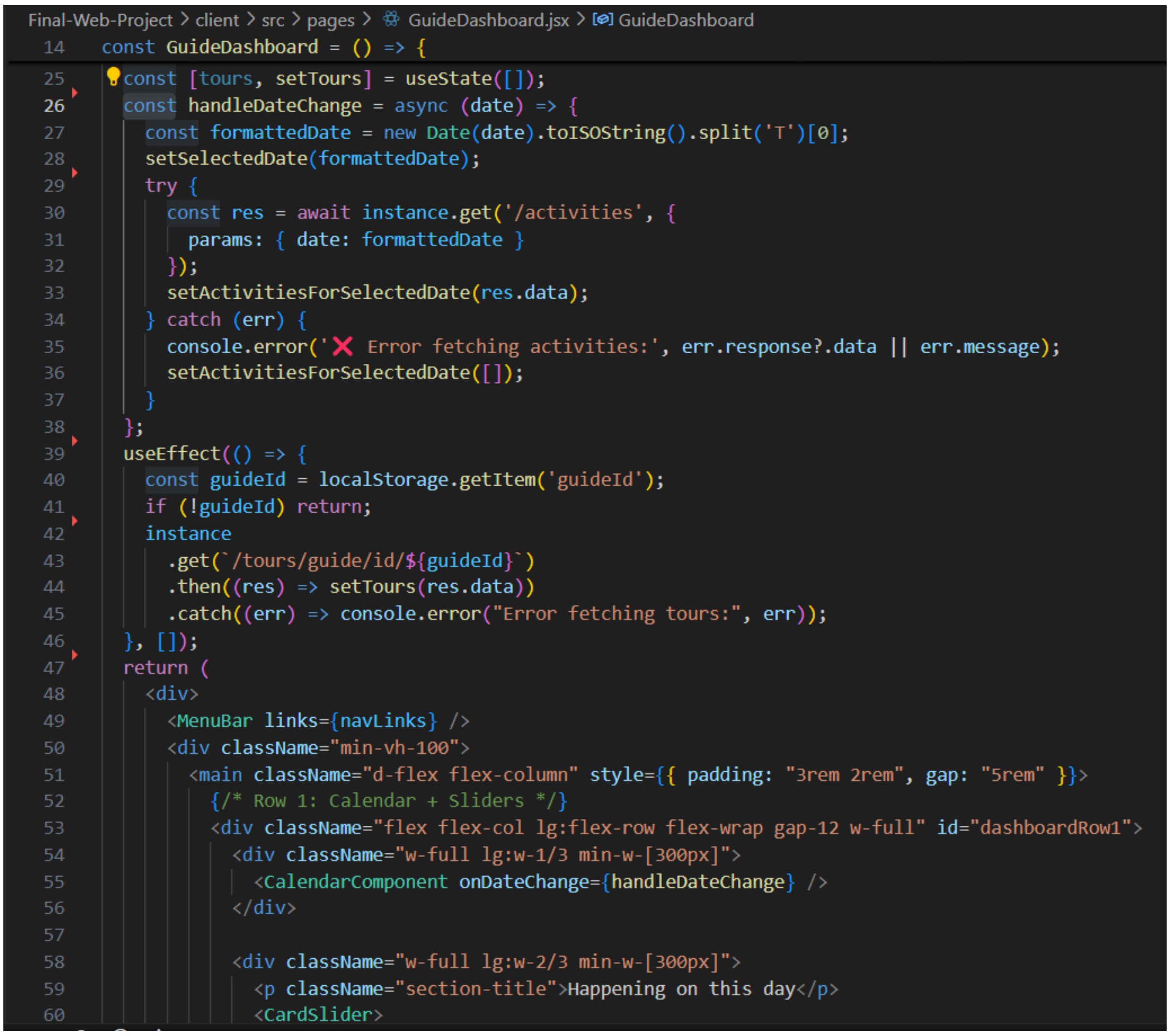
client > src > components > EarningPerMonth.jsx > EarningPerMonth > useEffect() callback > fe

```
const EarningPerMonth = () => {

  useEffect(() => {
    const fetchYears = async () => {
      try {
        const res = await axios.get('/earnings/years');
        setAvailableYears(res.data);
      } catch (err) {
        console.error('Error fetching years:', err);
      }
    };
    fetchYears();
  }, []);
  useEffect(() => {
    const fetchEarnings = async () => {
      if (!selectedYear) return;

      try {
        const res = await axios.get(`/earnings/${selectedYear}`);
        setEarnings(res.data);
      } catch (err) {
        console.error('Error fetching earnings:', err);
        setEarnings(new Array(12).fill(0));
      }
    };
    fetchEarnings();
  }, [selectedYear]);

  const chartData = months.map((month, index) => ({
    month,
    earning: earnings[index],
  }));

  return (
    <div className="earning-container">
      <div className="year-dropdown-centered">
```

The EarningPerMonth component is used to display a bar chart showing the tour guide's monthly earnings for a selected year. It provides a dropdown menu to choose the year and visualizes earnings using the recharts library.
When the component first loads, it sends a GET request to /earnings/years to fetch a list of years that have recorded earnings data (e.g., 2022, 2023). These values are shown in the dropdown selector for user input.

After a year is selected, the component makes another GET request to /earnings/:year (e.g., /earnings/2024). This returns an array of 12 numbers—each representing the earnings for one month of that year.
To render the chart, the component maps the array of values into a format compatible with Recharts. It creates a new object for each month that includes both the month name and its corresponding earning, which is then passed to the chart for display.
This combination of API integration, state management, and data transformation provides tour guides with a clear and interactive way to understand their monthly revenue trends.

8.



Final-Web-Project > client > src > pages > GuideDashboard.jsx > GuideDashboard

```jsx
const GuideDashboard = () => {
  const [tours, setTours] = useState([]);
  const handleDateChange = async (date) => {
    const formattedDate = new Date(date).toISOString().split('T')[0];
    setSelectedDate(formattedDate);
    try {
      const res = await instance.get('/activities', {
        params: { date: formattedDate }
      });
      setActivitiesForSelectedDate(res.data);
    } catch (err) {
      console.error('❌ Error fetching activities:', err.response?.data || err.message);
      setActivitiesForSelectedDate([]);
    }
  };
  useEffect(() => {
    const guideId = localStorage.getItem('guideId');
    if (!guideId) return;
    instance
      .get(`/tours/guide/id/${guideId}`)
      .then((res) => setTours(res.data))
      .catch((err) => console.error("Error fetching tours:", err));
  }, []);
  return (
    <div>
      <MenuBar links={navLinks} />
      <div className="min-vh-100">
        <main className="d-flex flex-column" style={{ padding: "3rem 2rem", gap: "5rem" }}>
          {/* Row 1: Calendar + Sliders */}
          <div className="flex flex-col lg:flex-row flex-wrap gap-12 w-full" id="dashboardRow1">
            <div className="w-full lg:w-1/3 min-w-[300px]">
              <CalendarComponent onDateChange={handleDateChange} />
            </div>

            <div className="w-full lg:w-2/3 min-w-[300px]">
              <p className="section-title">Happening on this day</p>
              <CardSlider>
```

This function listens for calendar input and fetches activities from the backend filtered by the selected date. It demonstrates API integration using query parameters and updates the frontend accordingly.
Retrieves the current guide's ID from localStorage and sends a GET request to fetch that guide’s tours. This ensures personalization and secure access to relevant content only.

Combines daily filtering (calendar) with monthly overview (slider). It displays activities based on user interaction and integrates multiple components dynamically.

e. Open Source Code Reusability:

The Jadwill project builds upon several open-source libraries and tools to facilitate its functionality and user interface. All of these tools have extensive usage in research and business environments and utilized according to their specific licenses. React.js, Node.js, Express.js, and Bootstrap were all used under the MIT License that allows free use, modification, and distribution with proper attribution. MongoDB Atlas, our cloud-hosted database, is based on the Server Side Public License (SSPL), which allows free use subject to certain conditions for cloud offerings. No proprietary software or paid libraries were incorporated within this project. All the components used were open-source and properly attributed when necessary.

f. Learnings and Reflections:

Key Learnings

1. **Project Planning & Task Distribution**
   - We began the project with a clear plan using tools like Google Sheets and Notion to divide responsibilities across all phases, we also assign roles to each team member
   - Each team member took ownership of specific components, ensuring parallel development and consistent integration that does not conflict with the rest.
   - Regular in person and online meetings and GitHub issues/commits were used to track progress and solve blockers collaboratively.

2. **Exploration of Technologies**
   - Although familiar with HTML and CSS, we learned and applied tools such as:
     - React.js for component-based UI development

        - Node.js & Express for RESTful backend APIs
        - MongoDB Atlas for cloud-based NoSQL storage
    - We explored deployment platforms like Render (backend) and frontend more than before, understanding their limitations and setup processes.

3. **API Integration**
    - Developed and tested several RESTful endpoints for activities, user data, etc.
    - Learned how to secure routes, handle errors, and use tools like Postman and Axios to simulate real user behavior.

4. **State & Data Management in React**
    - Gained experience with React hooks like useState, useEffect, and useParams to control component logic and routing.
    - Understood how to pass props, use conditional rendering, and fetch API data dynamically.

5. **Team Collaboration with Git**
    - Improved our skills with Git version control, managing merge conflicts and organizing code contributions through GitHub branches and pull requests.

6. **UI/UX Design Considerations**
    - Paid close attention to layout, card styling, and button feedback using Bootstrap 5 for better user experience.
    - We applied accessibility and mobile responsiveness.

Limitations (Updated)

1. **Free-Tier Infrastructure Constraints**
    - The project relied entirely on free-tier services such as:
        - Render for backend deployment

            - MongoDB Atlas (Free Cluster) for database hosting
    - These services have limited performance, restricted bandwidth, and may introduce cold-start delays, especially for backend APIs.

2. Limited Server Resources

- Due to free hosting plans, we experienced:
    - Occasional deployment timeouts
    - Slow data fetching when loading the dashboard or details

3. Time Constraints

- The project was completed within a fixed academic timeline, which limited the ability to:
    - Fully explore additional advanced features
    - Conduct thorough more detailed testing and optimization
    - Refactor and polish certain areas of the codebase

Future Improvements:

Upgrade from Free-Tier Hosting

- Move from Render (free) to a paid VPS or cloud provider (AWS, Heroku Pro, DigitalOcean)

Database Scaling and Optimization

- Upgrade from MongoDB Atlas free cluster to a dedicated cluster with:
    - Better storage and performance
    - Automated backups and monitoring
    - Support for horizontal scaling and indexing

Enhanced Deployment Pipeline

- Replace manual deployment with CI/CD integration using GitHub Actions or Netlify build hooks.

- Automate testing and deployment with each push to the main branch.

Integrated Payment Gateways

- Use third-party services like Stripe or PayPal for secure payment processing and booking monetization.

Third-Party Map Integration

- Replace static location displays with interactive Google Maps or Leaflet.js to enhance UX for activity locations and guide coverage.

g. Team Contributions (Time-sheet):

The Jadwill project was built through clear role-based collaboration across all phases. From **Phase 1 to Phase 6**, each team member was consistently responsible for all functionalities related to a specific user type. This included requirement gathering, prototyping, frontend and backend development. This structured division allowed each member to deeply understand and build their part of the system, ensuring high-quality and consistent output.

User Role Responsibilities (Phases 1–6)

- **Lamees & Nora – Tourist User**
 Responsible for all features and development related to the tourist flow, including:
  - Home, About, Where To?, Find A Local, My Plan, Wishlist
  - In addition, **Lamees and Nora** also designed and implemented the **public landing page (Home Page)** of the website, which introduces the platform before login.
- **Sarah & Aminah – Tour Guide User**
 Fully handled the tour guide experience from login to tour creation, including:
  - Profile, Tour Center, Dashboard
  - **Aminah** also implemented the **Login and Logout** system shared by all users.
  - **Sarah** led the **deployment** of the platform and handled related setup and testing.
- **Walah & Reem – Activity Provider User**
 Focused on building all functionality related to the activity provider, including:

- Provider Profile, Reservations, Events History
- In addition, **Walah and Reem** collaboratively wrote the **Final Report**, documenting the entire project structure, tools, roles, and learning outcomes.

Admin Panel and Shared Features

All six members collaboratively developed and contributed to the **Admin Panel**, which includes:

- Dashboard overview
- User Management
- Pending Activities
- Pending Registrations
- Complaints handling

Estimated Hours (per member)

Each member contributed **more than 50 hours** to the project, distributed across:

- Requirements & planning
- Design & development (frontend + backend)
- Testing & debugging
- Deployment
- Documentation & presentation

This structure ensured strong ownership, clear task distribution, and seamless integration across all system components.