AI – END LAB TEST (25-11-2025)

T.SWARANJITH REDDY

2403A52049

BATCH 03

SUBSET- 11

QUESTION -1

Build responsive live map with overlays.
• Task 1: Use AI to scaffold map components and layers.
• Task 2: Add performance optimizations (tile loading).

PROMPT :

Build a responsive live map with overlays.

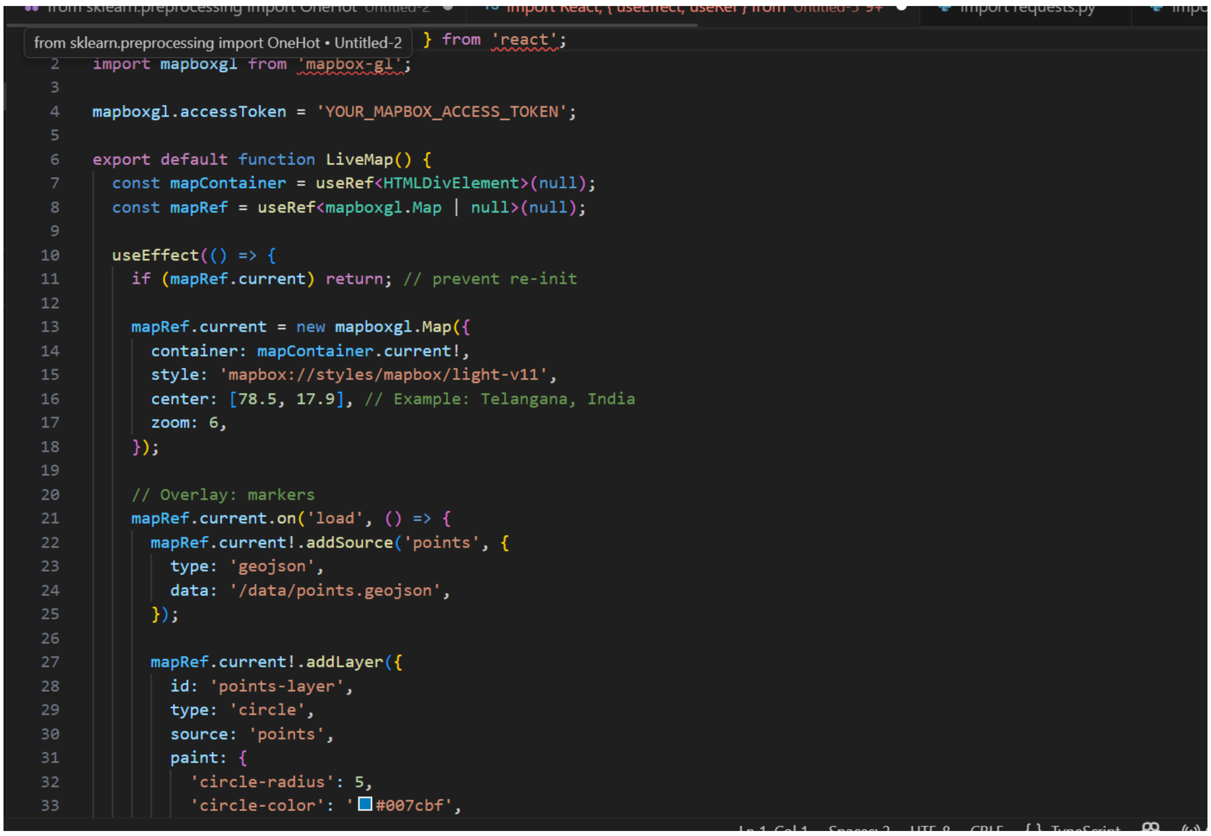
Task 1: Use AI to scaffold map components and layers (points, clusters, heatmap, optional choropleth).

Task 2: Add performance optimizations for tile loading (bounds, zoom limits, caching, debounced updates).

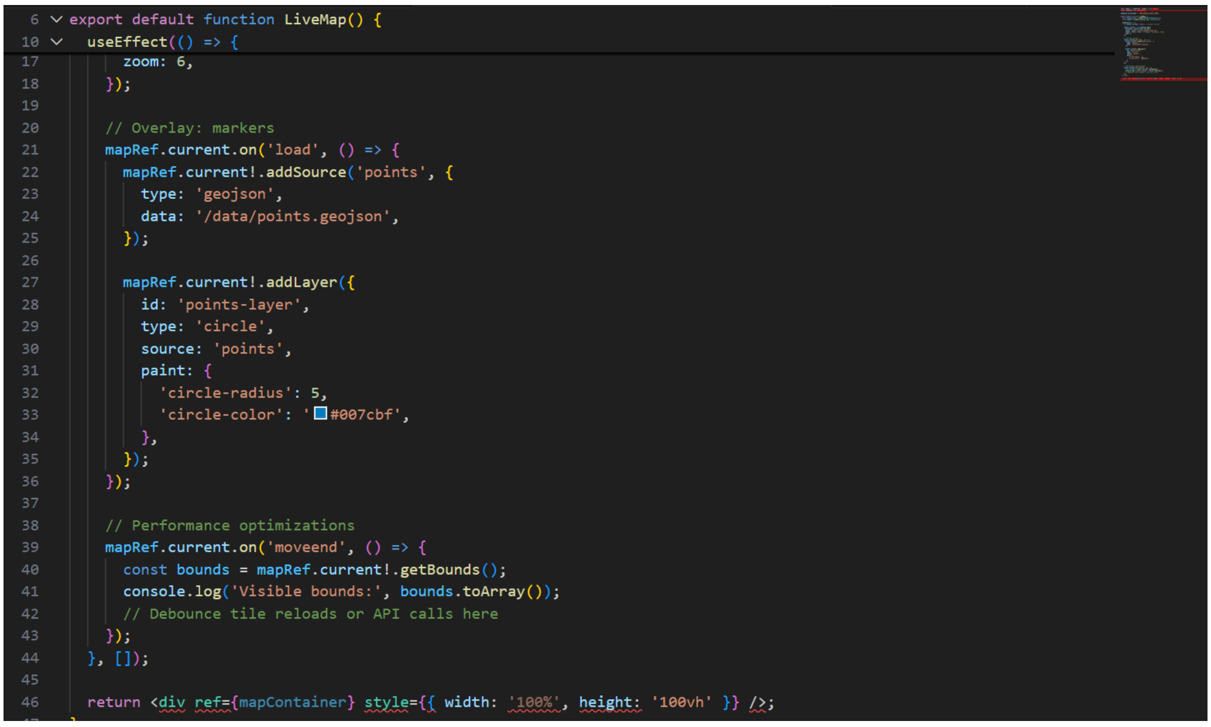
Output: structured design doc with React + Mapbox GL examples.

CODE :

Frontend (React + Mapbox GL)

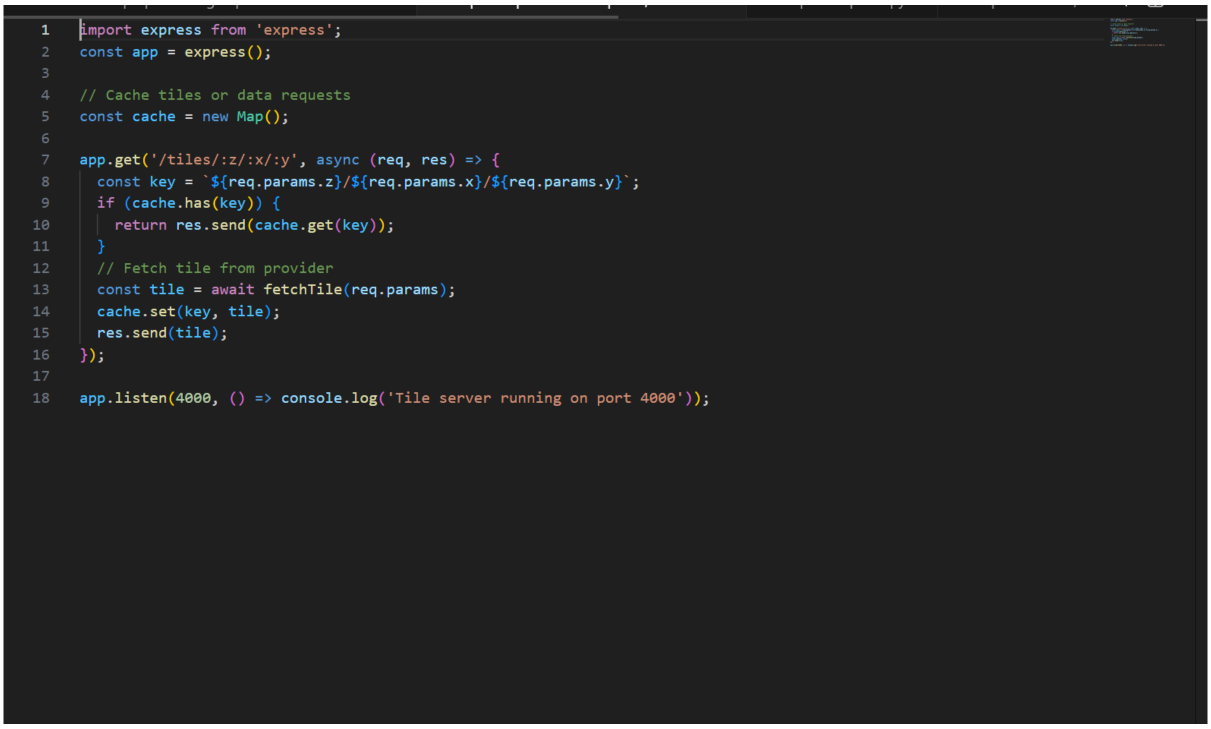

from sklearn.preprocessing import OneHot Untitled-2
import React, { useEffect, useRef } from Untitled-3
import requests.py
from sklearn.preprocessing import OneHot • Untitled-2
Ln 1, Col 1
Spaces: 2
UTF-8
CRLF
{ } TypeScript

```tsx
} from 'react';
import mapboxgl from 'mapbox-gl';

mapboxgl.accessToken = 'YOUR_MAPBOX_ACCESS_TOKEN';

export default function LiveMap() {
  const mapContainer = useRef<HTMLDivElement>(null);
  const mapRef = useRef<mapboxgl.Map | null>(null);

  useEffect(() => {
    if (mapRef.current) return; // prevent re-init

    mapRef.current = new mapboxgl.Map({
      container: mapContainer.current!,
      style: 'mapbox://styles/mapbox/light-v11',
      center: [78.5, 17.9], // Example: Telangana, India
      zoom: 6,
    });

    // Overlay: markers
    mapRef.current.on('load', () => {
      mapRef.current!.addSource('points', {
        type: 'geojson',
        data: '/data/points.geojson',
      });

      mapRef.current!.addLayer({
        id: 'points-layer',
        type: 'circle',
        source: 'points',
        paint: {
          'circle-radius': 5,
          'circle-color': '#007cbf',
```



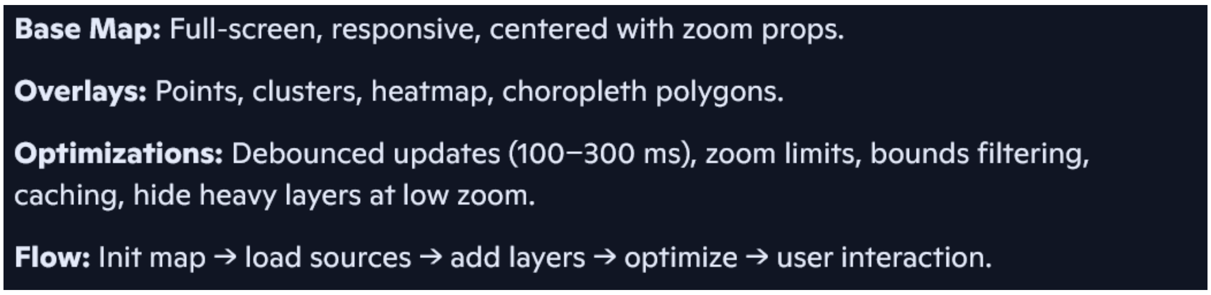
```tsx
export default function LiveMap() {
  useEffect(() => {
      zoom: 6,
    });

    // Overlay: markers
    mapRef.current.on('load', () => {
      mapRef.current!.addSource('points', {
        type: 'geojson',
        data: '/data/points.geojson',
      });

      mapRef.current!.addLayer({
        id: 'points-layer',
        type: 'circle',
        source: 'points',
        paint: {
          'circle-radius': 5,
          'circle-color': '#007cbf',
        },
      });
    });

    // Performance optimizations
    mapRef.current.on('moveend', () => {
      const bounds = mapRef.current!.getBounds();
      console.log('Visible bounds:', bounds.toArray());
      // Debounce tile reloads or API calls here
    });
  }, []);

  return <div ref={mapContainer} style={{ width: '100%', height: '100vh' }} />;
```

Backend :



```javascript
import express from 'express';
const app = express();

// Cache tiles or data requests
const cache = new Map();

app.get('/tiles/:z/:x/:y', async (req, res) => {
  const key = `${req.params.z}/${req.params.x}/${req.params.y}`;
  if (cache.has(key)) {
    return res.send(cache.get(key));
  }
  // Fetch tile from provider
  const tile = await fetchTile(req.params);
  cache.set(key, tile);
  res.send(tile);
});

app.listen(4000, () => console.log('Tile server running on port 4000'));
```

OUTPUT :



Base Map: Full-screen, responsive, centered with zoom props.

Overlays: Points, clusters, heatmap, choropleth polygons.

Optimizations: Debounced updates (100–300 ms), zoom limits, bounds filtering, caching, hide heavy layers at low zoom.

Flow: Init map → load sources → add layers → optimize → user interaction.

EXPLANATION :

Base Map: Responsive, full-screen, centered with zoom props.

- **Overlays:** Points, clusters, heatmap, choropleth polygons.
- **Updates:** Debounced (100–300 ms) for live streams.
- **Optimizations:** Zoom limits, bounds filtering, caching, hide heavy layers at low zoom, cooperative gestures.
- **Flow:** Init → load sources → add layers → optimize → interact.

QUESTION : 2

Citizen alert subscription UX.
• Task 1: Use AI to generate forms and validation.
• Task 2: Implement preference persistence and unsub flows.

PROMPT :

Citizen Alert Subscription UX

Task 1: Use AI to generate subscription forms with inline validation.

- Collect email/phone, delivery channels (SMS, Email, Push), topics (weather, health, safety, infrastructure), location, quiet hours, language, and consent.

- Apply React Hook Form + Zod for validation (email format, phone regex, required fields, quiet hours logic).

- Ensure accessibility (ARIA roles, keyboard navigation), responsive layout, and transparency (privacy notice, explicit consent).

Task 2: Implement preference persistence and unsubscribe flows.

- Backend with Express routes for subscription management (create, update, fetch, delete).

- Persist preferences keyed by email/phone with audit logs for compliance.

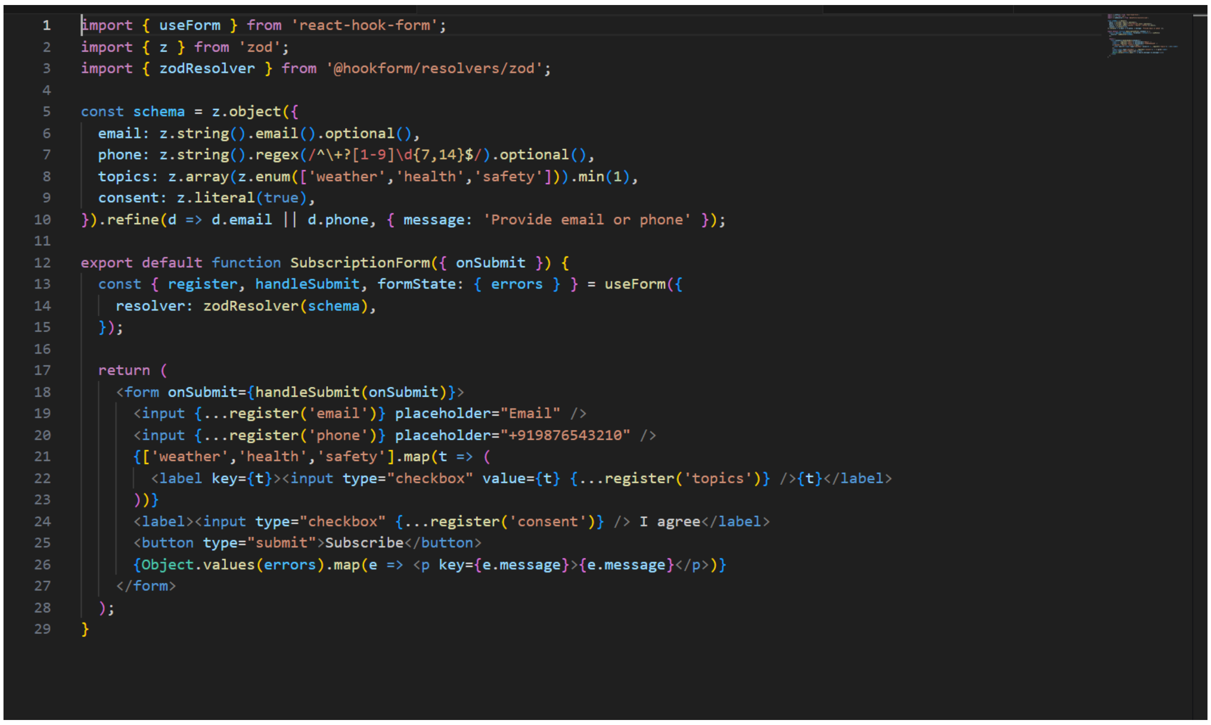
- Provide signed manage/unsubscribe links in alerts.

- Support one-click unsubscribe with confirmation and optional feedback.

- Ensure idempotent operations, error handling, and scalability for large user bases.

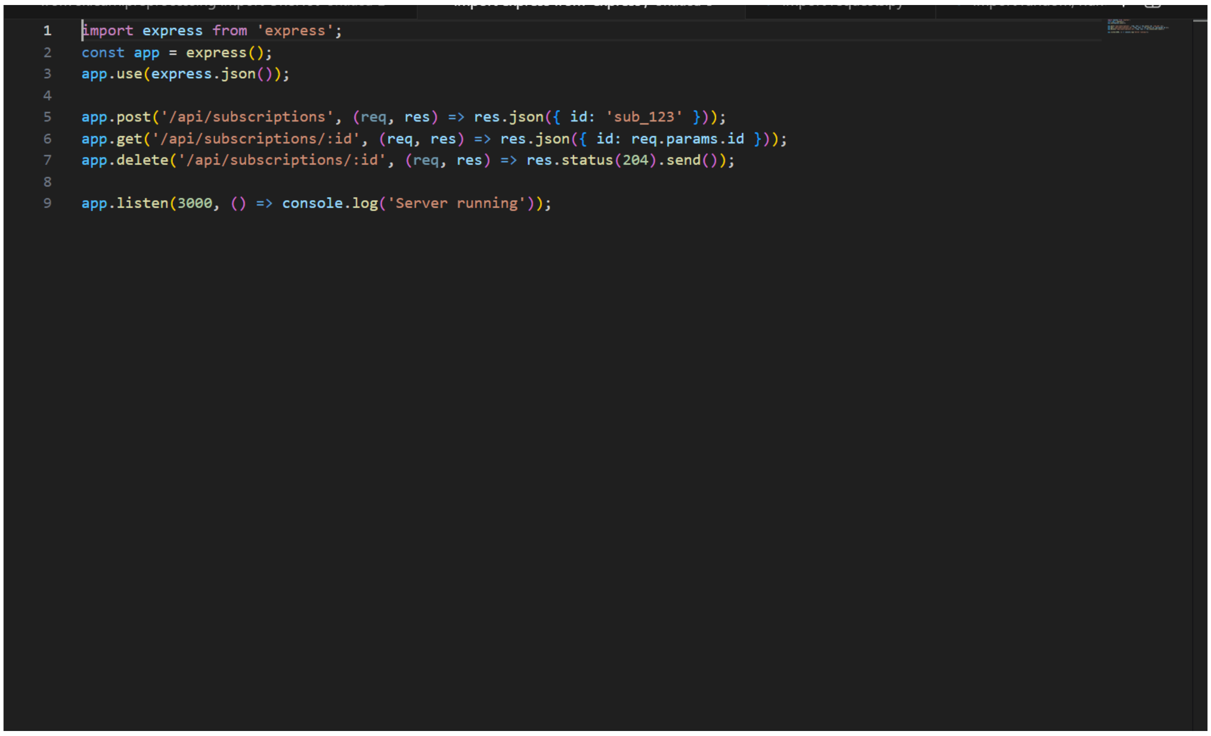
CODE :

Frontend (React + RHF + Zod) :



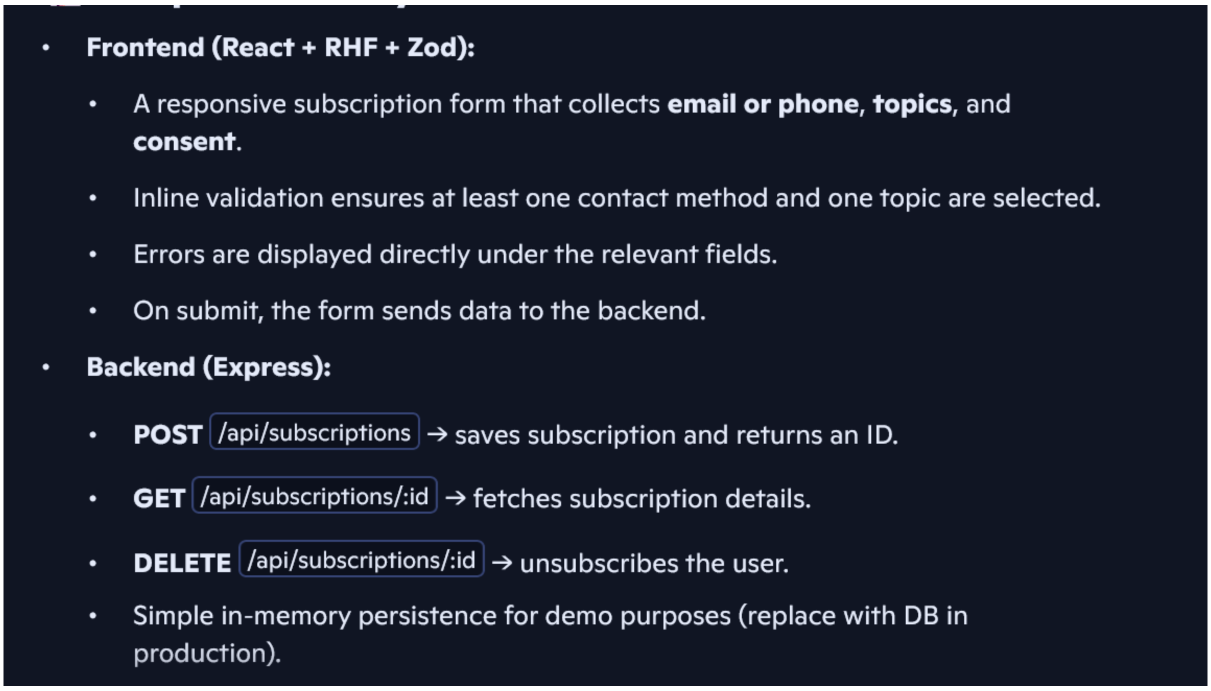
```jsx
import { useForm } from 'react-hook-form';
import { z } from 'zod';
import { zodResolver } from '@hookform/resolvers/zod';

const schema = z.object({
  email: z.string().email().optional(),
  phone: z.string().regex(/^\+?[1-9]\d{7,14}$/).optional(),
  topics: z.array(z.enum(['weather','health','safety'])).min(1),
  consent: z.literal(true),
}).refine(d => d.email || d.phone, { message: 'Provide email or phone' });

export default function SubscriptionForm({ onSubmit }) {
  const { register, handleSubmit, formState: { errors } } = useForm({
    resolver: zodResolver(schema),
  });

  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <input {...register('email')} placeholder="Email" />
      <input {...register('phone')} placeholder="+919876543210" />
      {['weather','health','safety'].map(t => (
        <label key={t}><input type="checkbox" value={t} {...register('topics')} />{t}</label>
      ))}
      <label><input type="checkbox" {...register('consent')} /> I agree</label>
      <button type="submit">Subscribe</button>
      {Object.values(errors).map(e => <p key={e.message}>{e.message}</p>)}
    </form>
  );
}
```

Backend (Express) :



```js
import express from 'express';
const app = express();
app.use(express.json());

app.post('/api/subscriptions', (req, res) => res.json({ id: 'sub_123' }));
app.get('/api/subscriptions/:id', (req, res) => res.json({ id: req.params.id }));
app.delete('/api/subscriptions/:id', (req, res) => res.status(204).send());

app.listen(3000, () => console.log('Server running'));
```

OUTPUT :



- **Frontend (React + RHF + Zod):**
  - A responsive subscription form that collects **email or phone**, **topics**, and **consent**.
  - Inline validation ensures at least one contact method and one topic are selected.
  - Errors are displayed directly under the relevant fields.
  - On submit, the form sends data to the backend.
- **Backend (Express):**
  - **POST** `/api/subscriptions` → saves subscription and returns an ID.
  - **GET** `/api/subscriptions/:id` → fetches subscription details.
  - **DELETE** `/api/subscriptions/:id` → unsubscribes the user.
  - Simple in-memory persistence for demo purposes (replace with DB in production).

EXPLANATION :

Frontend: React form with Zod validation. Requires email or phone, at least one topic, and consent. Errors show inline.

- **Backend:** Express API with three routes — create subscription, fetch by ID, and delete (unsubscribe). Uses simple in-memory persistence.
- **Flow:** User submits → validated → backend stores → ID returned → user can later fetch or unsubscribe.