**DBMS PROJECT REPORT**

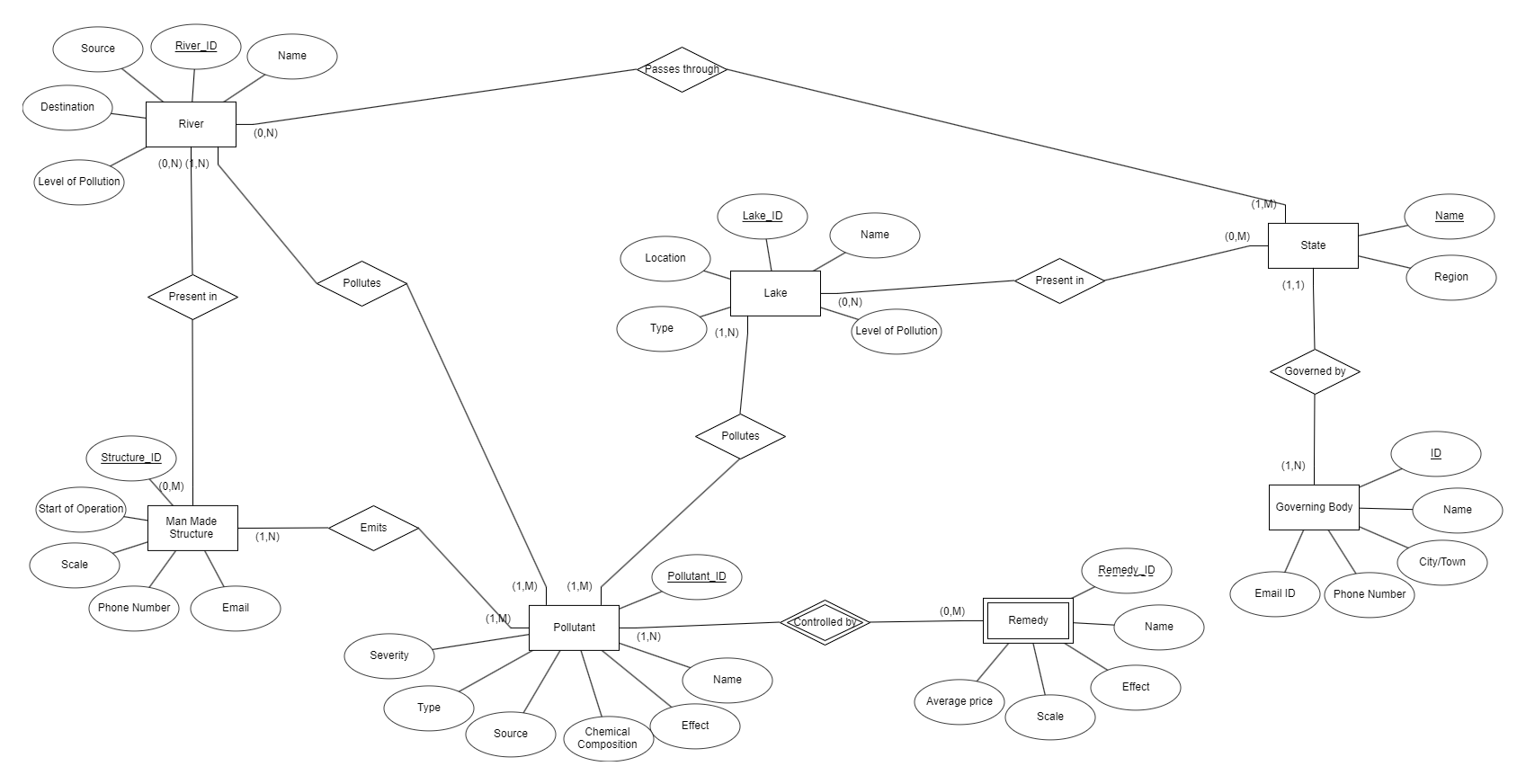
**TEAM** :-

UTKARSH SETH - PES1UG21CS687

VIBHA MARPALLE - PES1UG21CS708

**WATER POLLUTION MANAGEMENT DATABASE**

* **Description** : The Water Pollution Management Database Project aims to create a comprehensive and efficient system for monitoring, managing, and mitigating water pollution. This project stores state wise information about water bodies like pollution levels, pollutants and remedies which can be updated by government users and has a feature to raise requests regarding a body by citizen users.
* **List of Software used** : MySQL, Javascript, HTML, CSS, ReactJS, NodeJS, ExpressJS.
* **ER Diagram**



* **Relational Schema**



* **DDL SQL commands**

CREATE DATABASE miniProject;

USE miniProject;

CREATE TABLE rivers (

`river\_id` VARCHAR(10) PRIMARY KEY,

`name` VARCHAR(30) NOT NULL,

`level\_of\_pollution` ENUM("1", "2", "3", "4", "5"),

`source` VARCHAR(30),

`destination` VARCHAR(30)

);

CREATE TABLE states (

`name` VARCHAR(30) PRIMARY KEY,

`region` VARCHAR(30) NOT NULL

);

CREATE TABLE lakes (

`lake\_id` VARCHAR(10) PRIMARY KEY,

`name` VARCHAR(30) NOT NULL,

`level\_of\_pollution` ENUM("1", "2", "3", "4", "5"),

`location` VARCHAR(50),

`type` ENUM("Natural", "Man-made"),

`state\_name` VARCHAR(30)

);

ALTER TABLE lakes ADD FOREIGN KEY (`state\_name`) REFERENCES states(`name`) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE pollutants (

`pollutant\_id` VARCHAR(10) PRIMARY KEY,

`name` VARCHAR(30) NOT NULL,

`chemical\_composition` VARCHAR(30),

`source` VARCHAR(30),

`type` VARCHAR(30),

`severity` ENUM("High", "Medium", "Low")

);

CREATE TABLE man\_made\_structures (

`structure\_id` VARCHAR(10) PRIMARY KEY,

`start\_of\_operation` DATE NOT NULL,

`scale` ENUM("Small", "Medium", "Large") NOT NULL,

`phone\_number` VARCHAR(15) NOT NULL,

`email` VARCHAR(50) NOT NULL,

`river\_id` VARCHAR(10) NOT NULL,

CONSTRAINT FOREIGN KEY (`river\_id`) REFERENCES rivers(`river\_id`) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE river\_pollutants (

`river\_id` VARCHAR(10),

`pollutant\_id` VARCHAR(10),

`concentration` VARCHAR(20) NOT NULL,

`effect` VARCHAR(100) NOT NULL,

CONSTRAINT PRIMARY KEY (`river\_id`, `pollutant\_id`),

CONSTRAINT FOREIGN KEY (`river\_id`) REFERENCES rivers(`river\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FOREIGN KEY (`pollutant\_id`) REFERENCES pollutants(`pollutant\_id`) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE lake\_pollutants (

`lake\_id` VARCHAR(10),

`pollutant\_id` VARCHAR(10),

`concentration` VARCHAR(20) NOT NULL,

`effect` VARCHAR(100) NOT NULL,

CONSTRAINT PRIMARY KEY (`lake\_id`, `pollutant\_id`),

CONSTRAINT FOREIGN KEY (`lake\_id`) REFERENCES lakes(`lake\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FOREIGN KEY (`pollutant\_id`) REFERENCES pollutants(`pollutant\_id`) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE rivers\_in\_state (

`river\_id` VARCHAR(10),

`state\_name` VARCHAR(30),

CONSTRAINT PRIMARY KEY (`river\_id`, `state\_name`),

CONSTRAINT FOREIGN KEY (`river\_id`) REFERENCES rivers(`river\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FOREIGN KEY (`state\_name`) REFERENCES states(`name`) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE pollutants\_released (

`structure\_id` VARCHAR(10),

`pollutant\_id` VARCHAR(10),

`amount\_released` VARCHAR(30) NOT NULL,

CONSTRAINT PRIMARY KEY (`structure\_id`, `pollutant\_id`),

CONSTRAINT FOREIGN KEY (`structure\_id`) REFERENCES man\_made\_structures(`structure\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FOREIGN KEY (`pollutant\_id`) REFERENCES pollutants(`pollutant\_id`) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE government\_bodies (

`body\_id` VARCHAR(10),

`name` VARCHAR(45) NOT NULL,

`city/town` VARCHAR(45) NOT NULL,

`phone` VARCHAR(15) NOT NULL,

`email` VARCHAR(45) NULL,

`state\_name` VARCHAR(30) NOT NULL,

PRIMARY KEY (`body\_id`)

);

ALTER TABLE government\_bodies ADD CONSTRAINT FOREIGN KEY (`state\_name`) REFERENCES states(`name`) ON DELETE CASCADE ON UPDATE CASCADE;

CREATE TABLE remedies (

`pollutant\_id` VARCHAR(10),

`remedy\_id` VARCHAR(10) NOT NULL,

`name` VARCHAR(45) NOT NULL,

`effect` VARCHAR(100) NOT NULL,

`scale` VARCHAR(45) NULL,

`avg\_price` FLOAT NOT NULL,

PRIMARY KEY (pollutant\_id, remedy\_id),

CONSTRAINT FOREIGN KEY (`pollutant\_id`) REFERENCES pollutants(`pollutant\_id`) ON DELETE CASCADE ON UPDATE CASCADE

);

CREATE TABLE users (

`email` VARCHAR(50) PRIMARY KEY,

`first\_name` VARCHAR(30) NOT NULL,

`last\_name` VARCHAR(30) NOT NULL,

`password` VARCHAR(50) NOT NULL

);

CREATE TABLE admins (

`id` INT PRIMARY KEY,

`first\_name` VARCHAR(30) NOT NULL,

`last\_name` VARCHAR(30) NOT NULL,

`password` VARCHAR(30) NOT NULL

);

CREATE TABLE government\_users (

`government\_id` INT PRIMARY KEY,

`first\_name` VARCHAR(30) NOT NULL,

`last\_name` VARCHAR(30) NOT NULL,

`email` VARCHAR(50) NOT NULL,

`password` VARCHAR(50) NOT NULL

);

CREATE TABLE requests (

`request\_id` VARCHAR(15) PRIMARY KEY,

`user\_email` VARCHAR(50),

`lake\_id` VARCHAR(10),

`river\_id` VARCHAR(10),

`city` VARCHAR(45) NOT NULL,

`state\_name` VARCHAR(30),

`content` VARCHAR(300) NOT NULL,

CONSTRAINT FOREIGN KEY (`user\_email`) REFERENCES users(`email`) ON DELETE CASCADE ON UPDATE CASCADE,

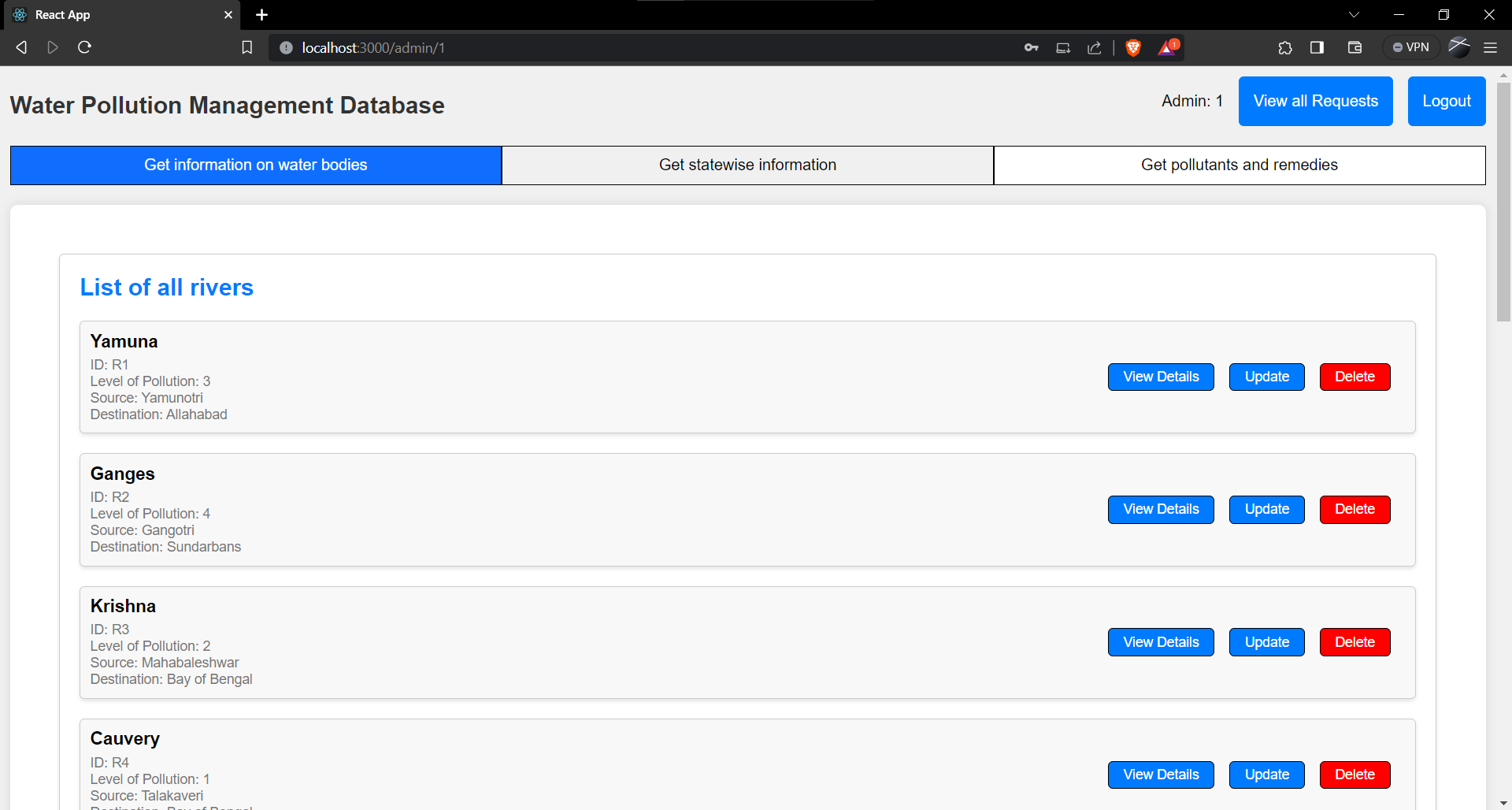
CONSTRAINT FOREIGN KEY (`lake\_id`) REFERENCES lakes(`lake\_id`) ON DELETE CASCADE ON UPDATE CASCADE,

CONSTRAINT FOREIGN KEY (`state\_name`) REFERENCES states(`name`) ON DELETE CASCADE ON UPDATE CASCADE

);

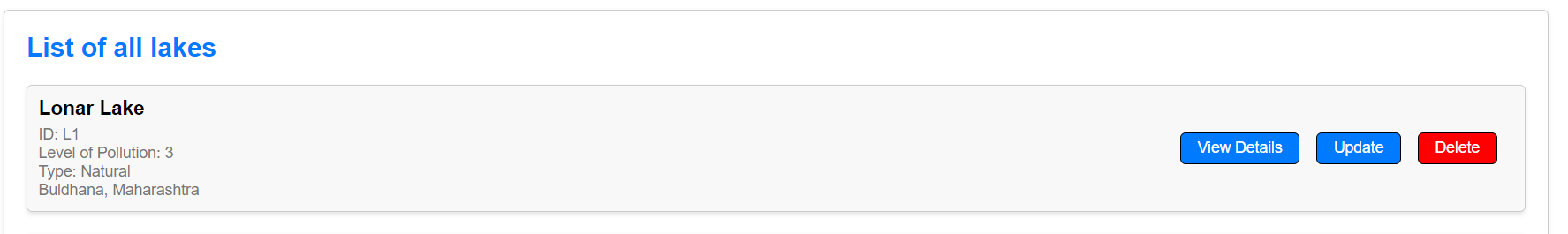
* **CRUD Operations Screenshots**

1) Read

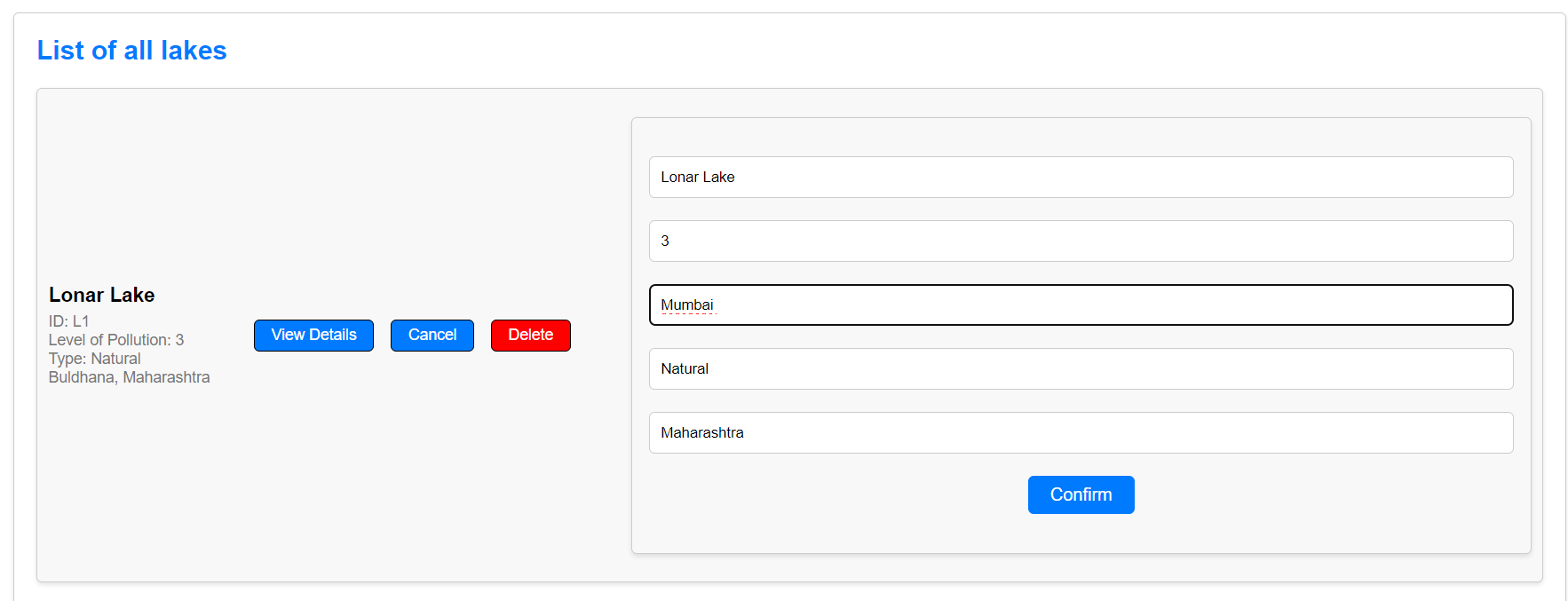


2) Update

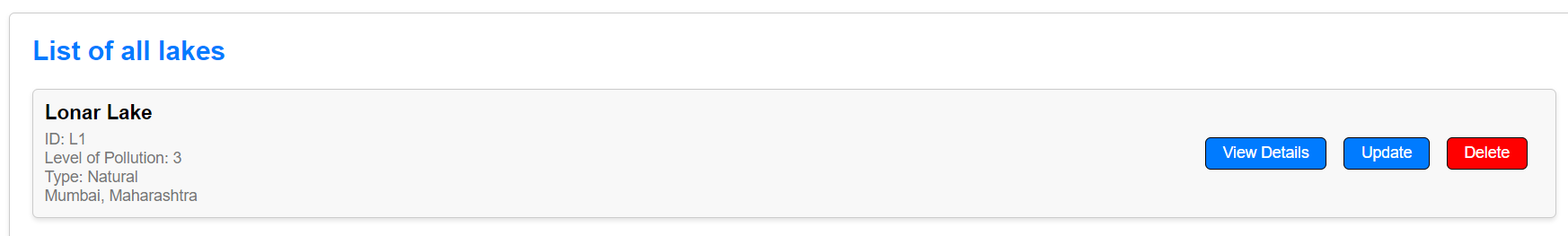
Before:



Updating the city:

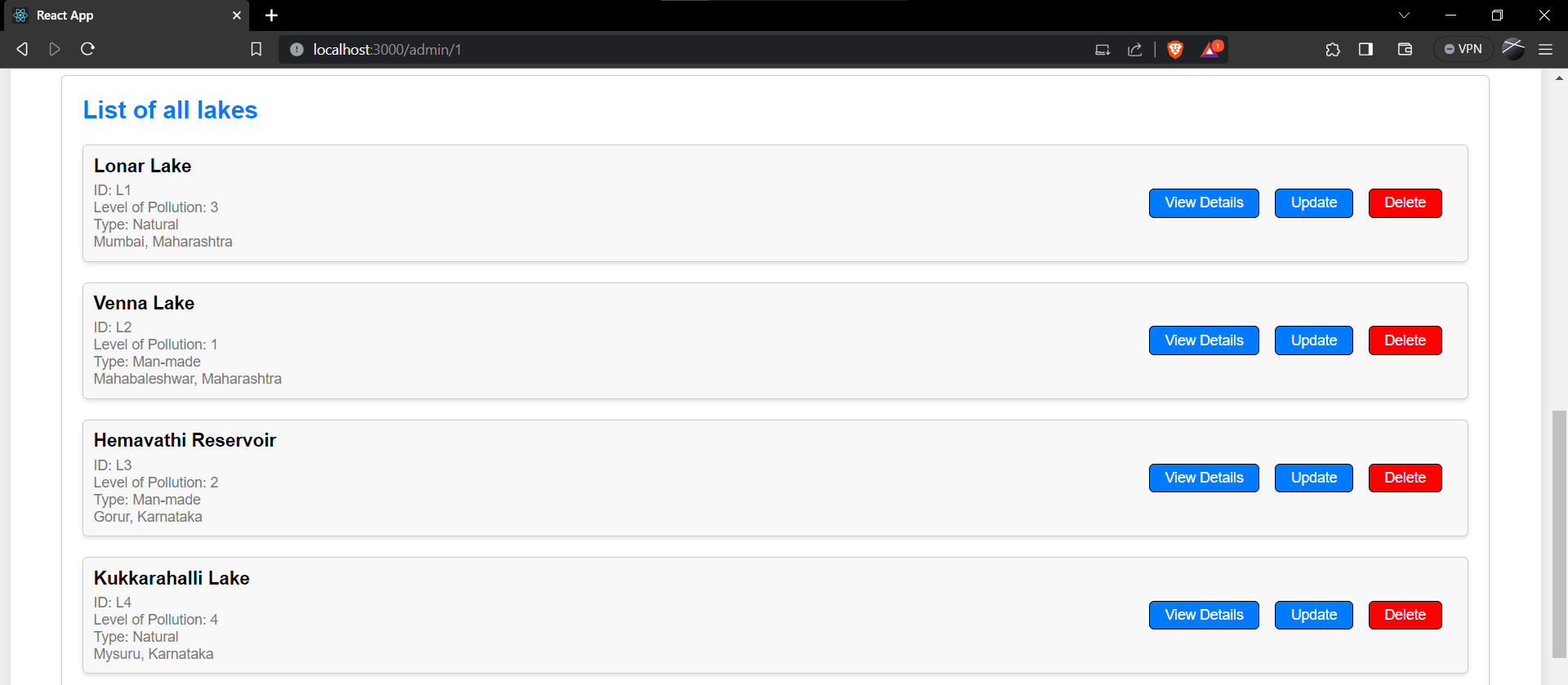


After:

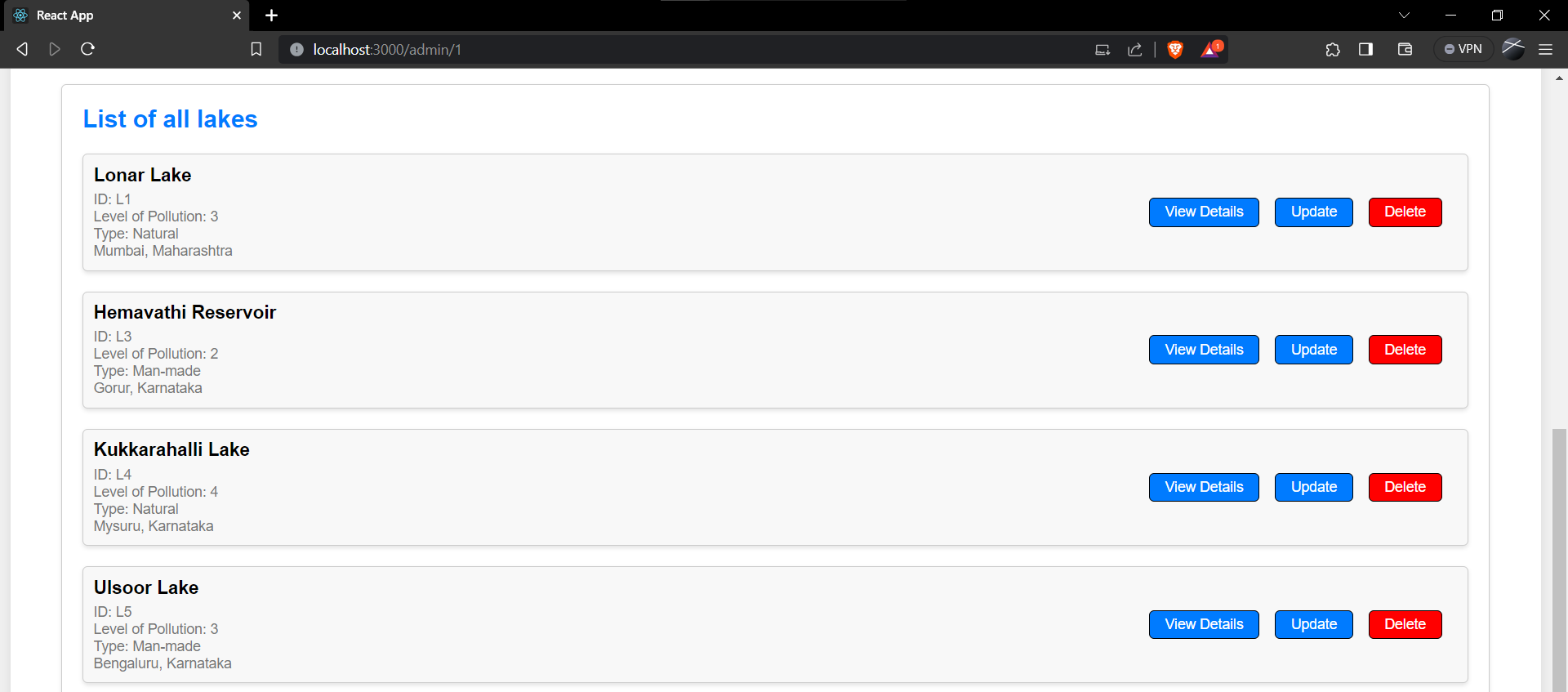


3) Delete

Before:



After deleting Venna Lake:

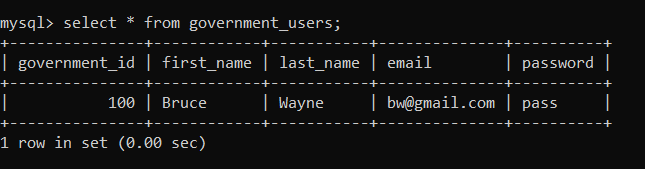


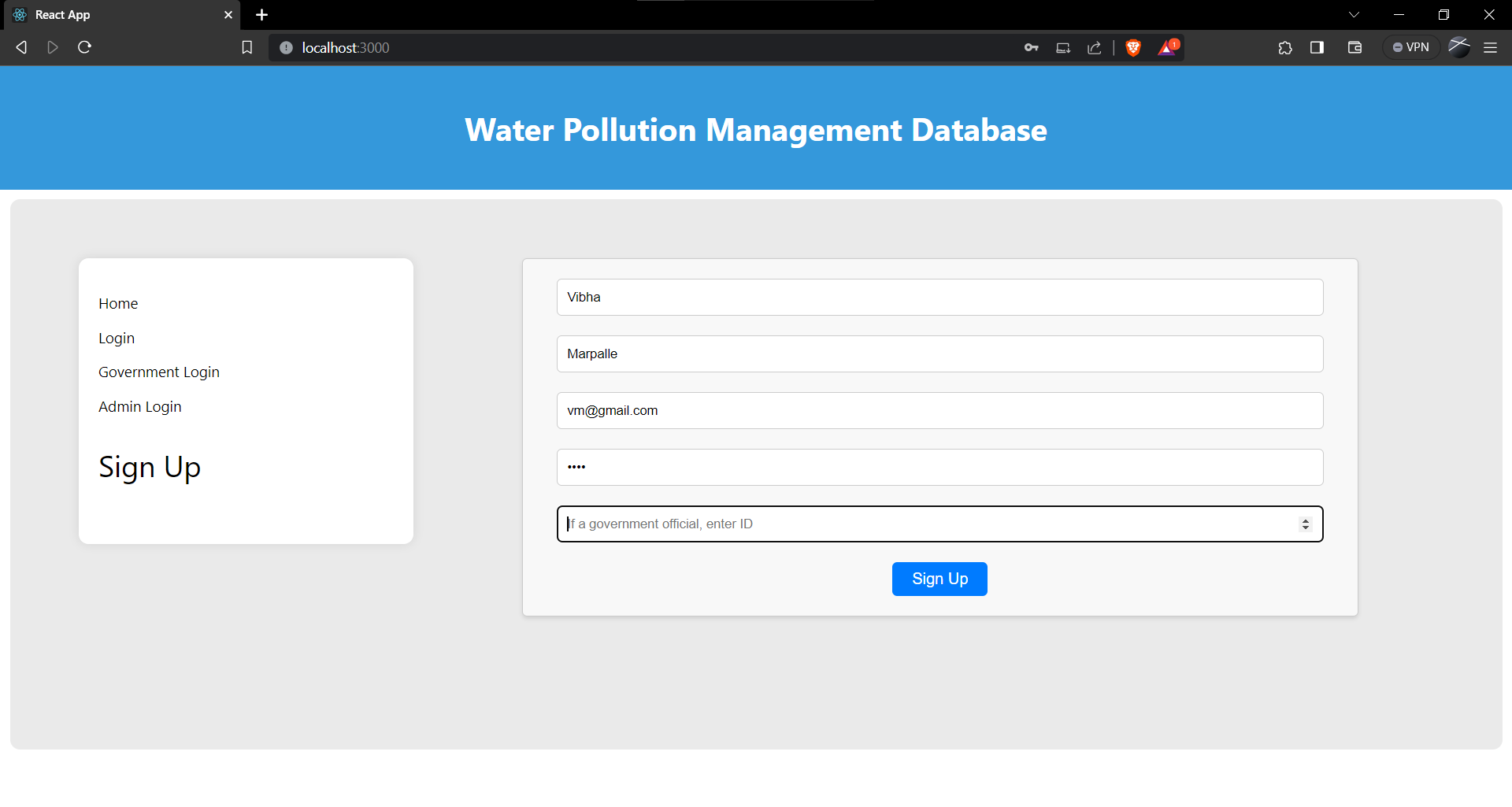
* **List of Functionalities with frontend screenshot**

1) Signup/Login for government bodies

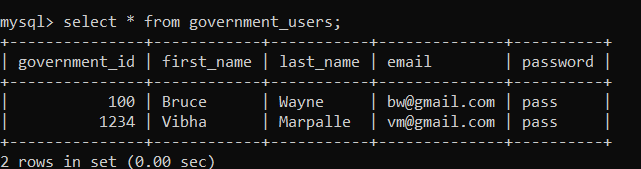
a) Signup

Before:

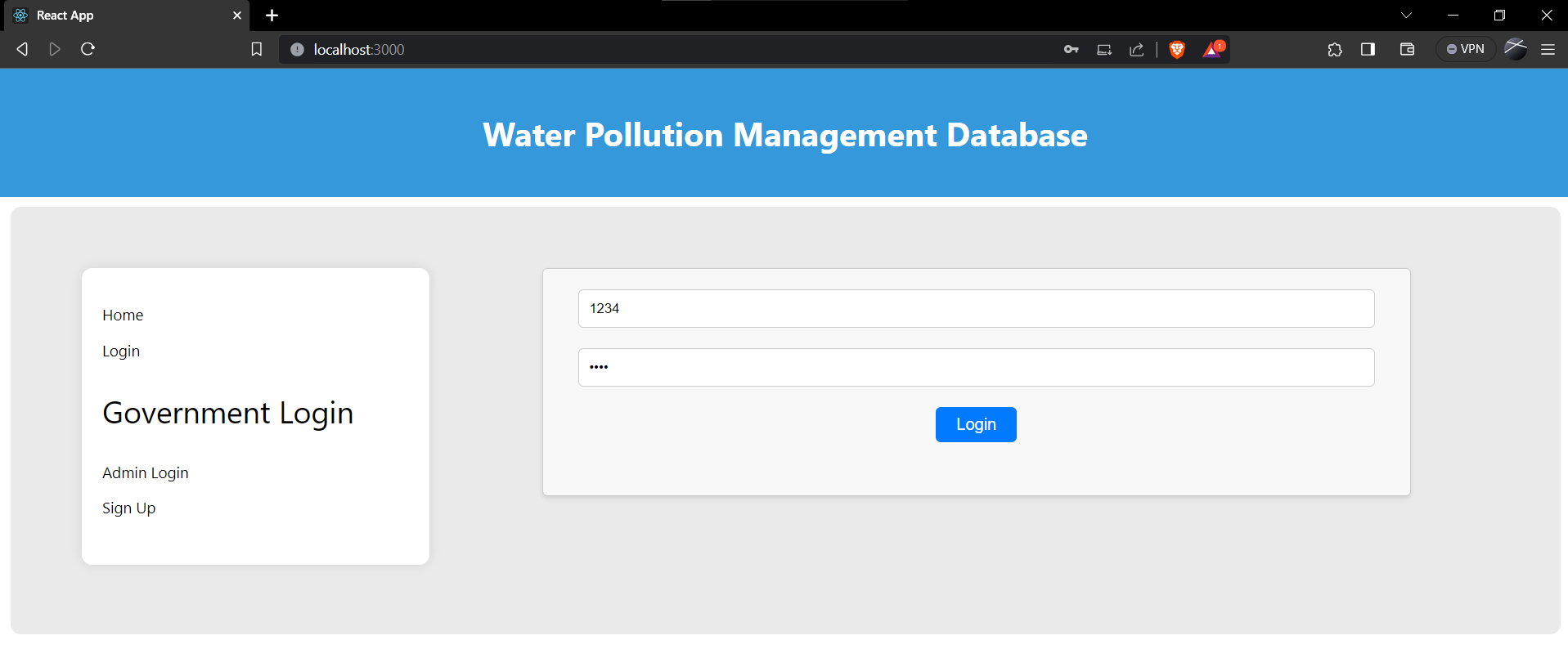


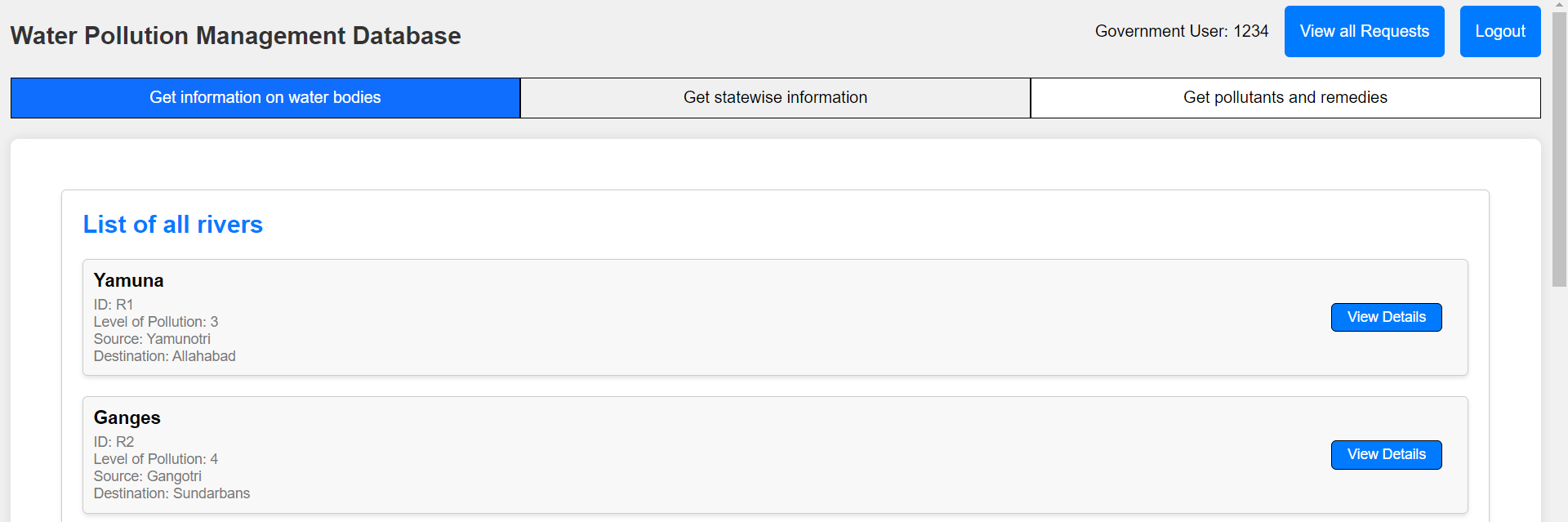


After:

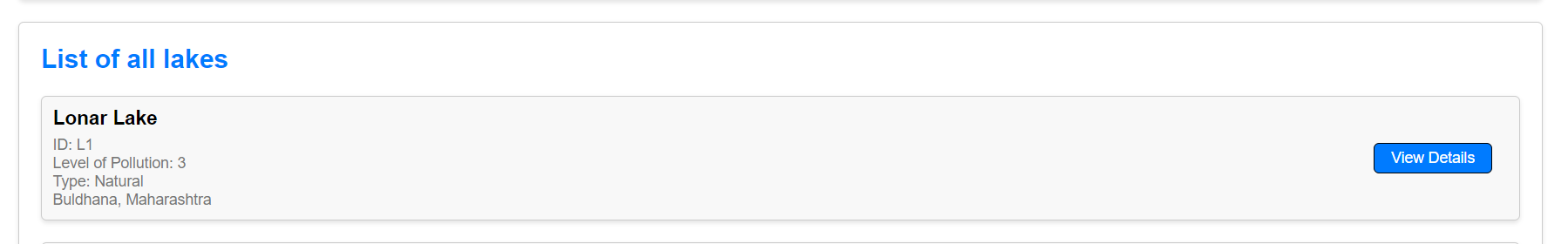


Login:

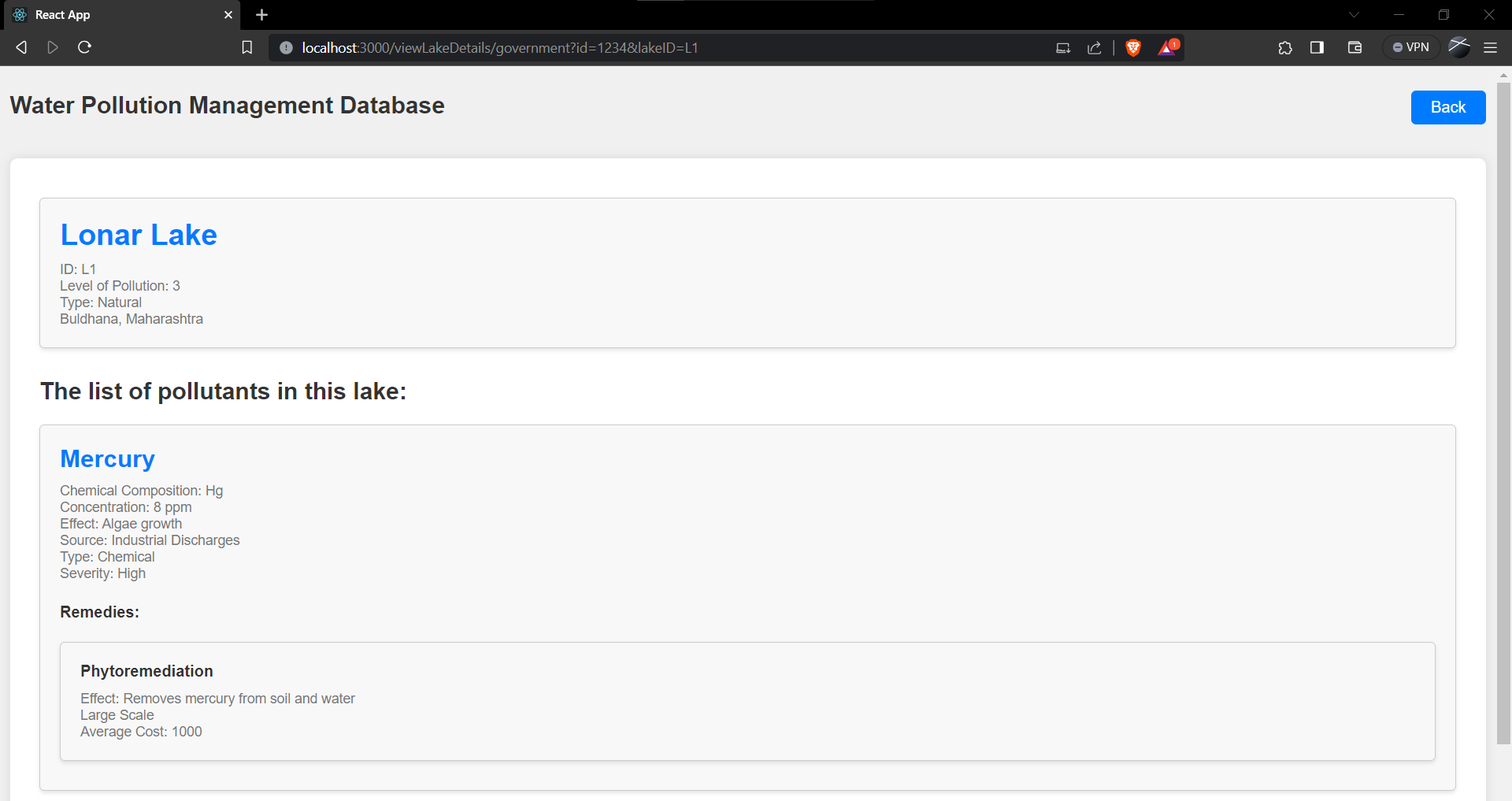




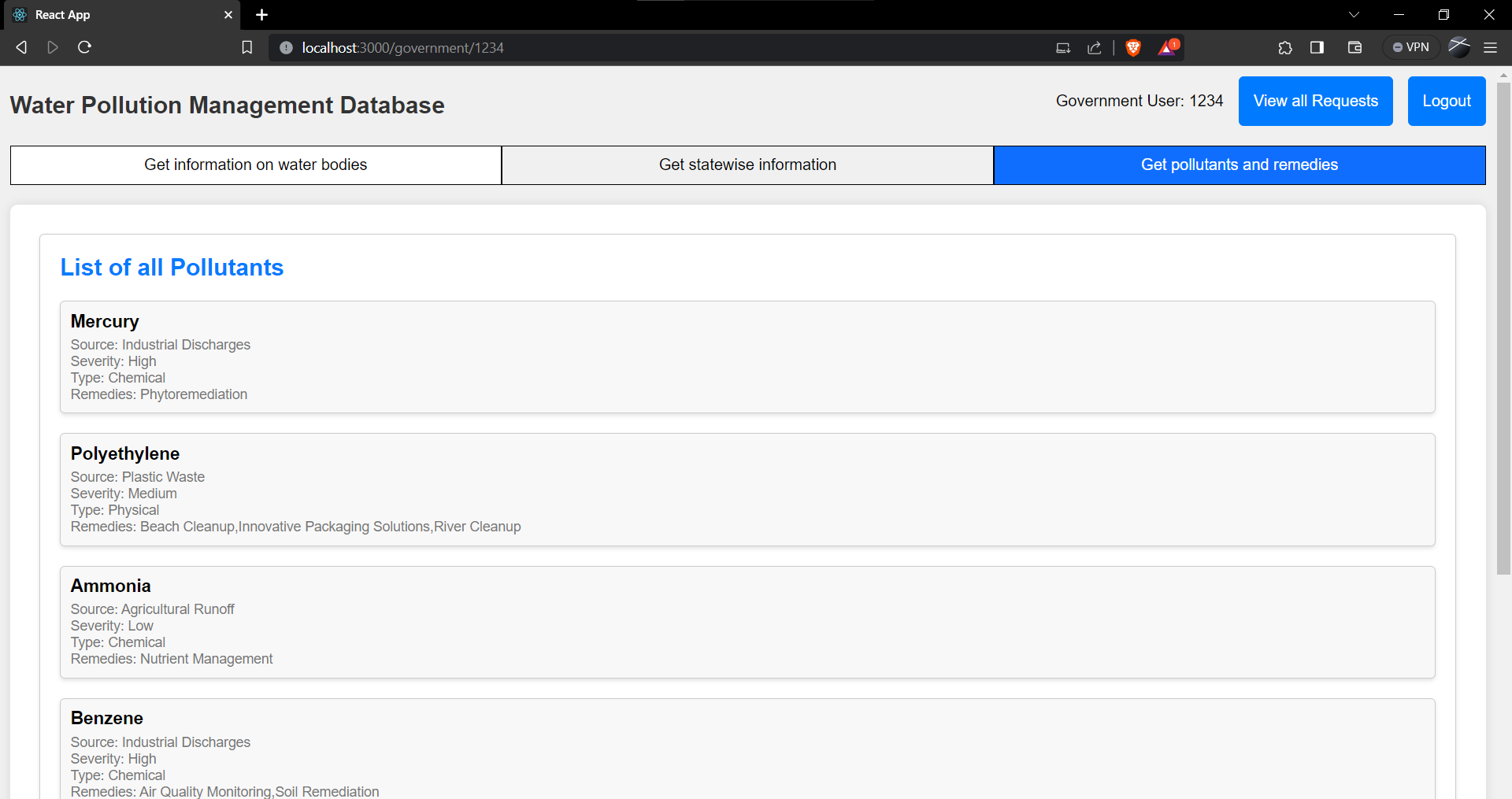
2) Viewing the pollution status for a water body



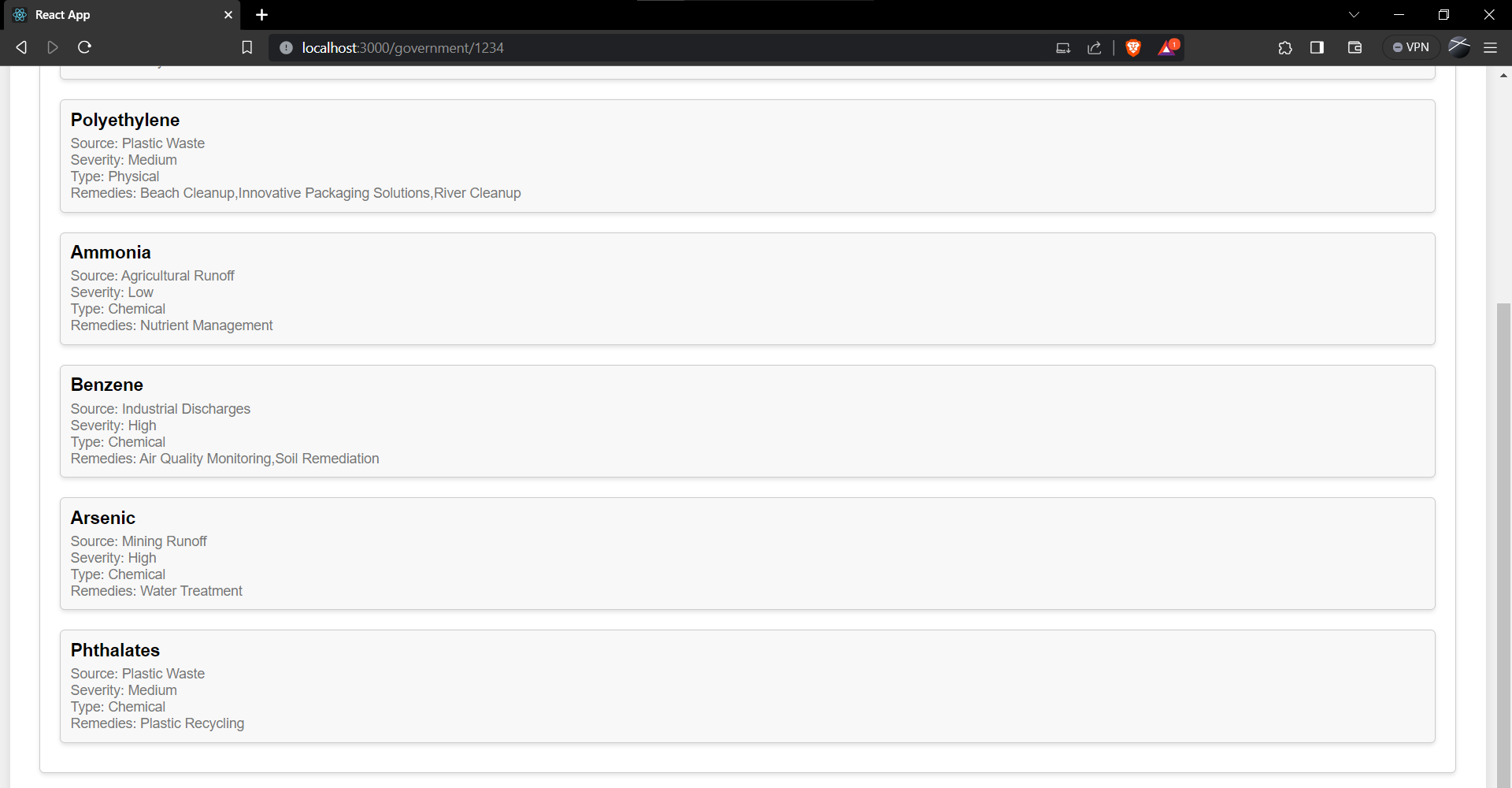
After clicking on ‘View Details’:

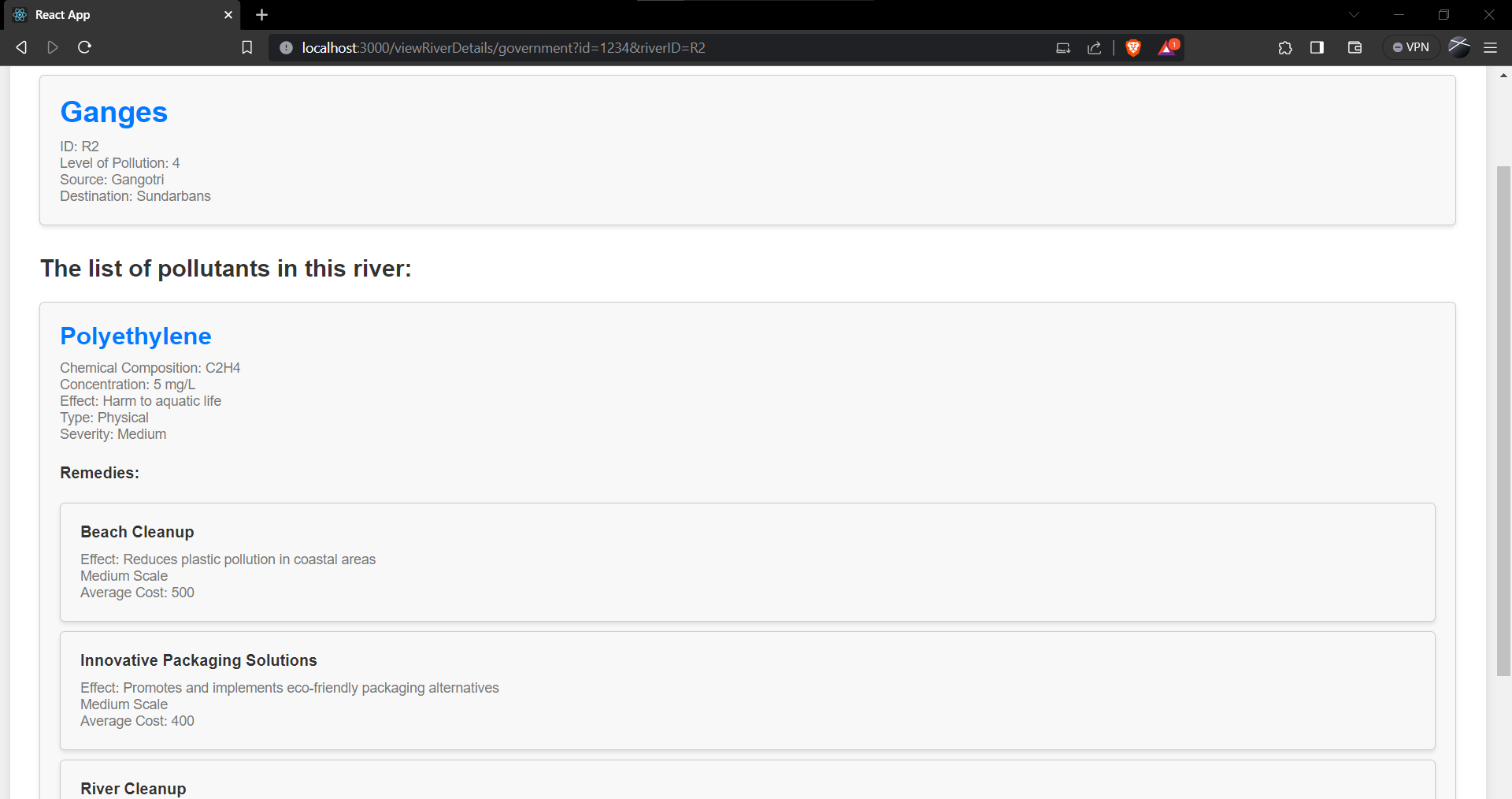


3) Finding the source of pollution



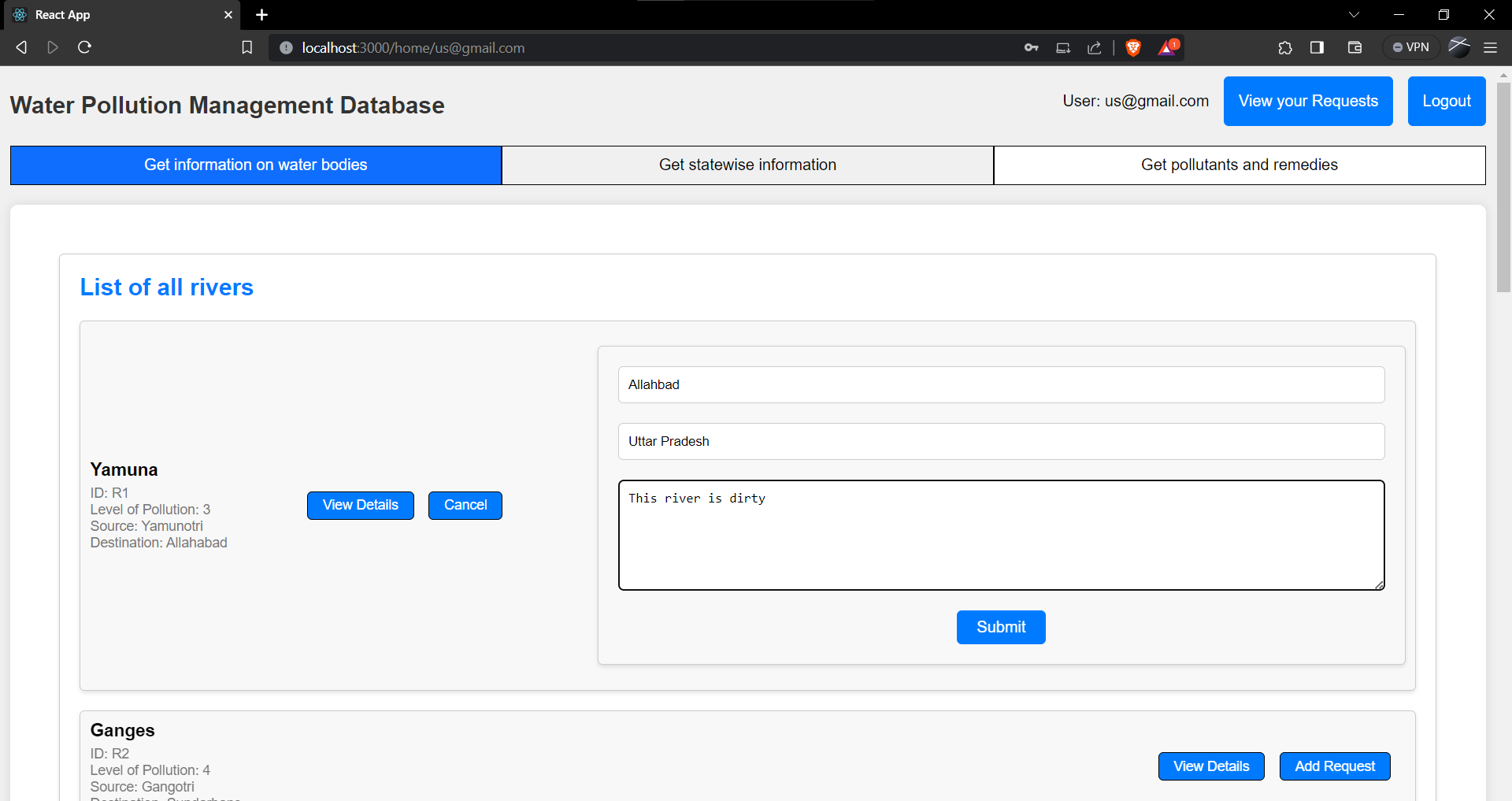
4) Giving remedies for a type of pollutant



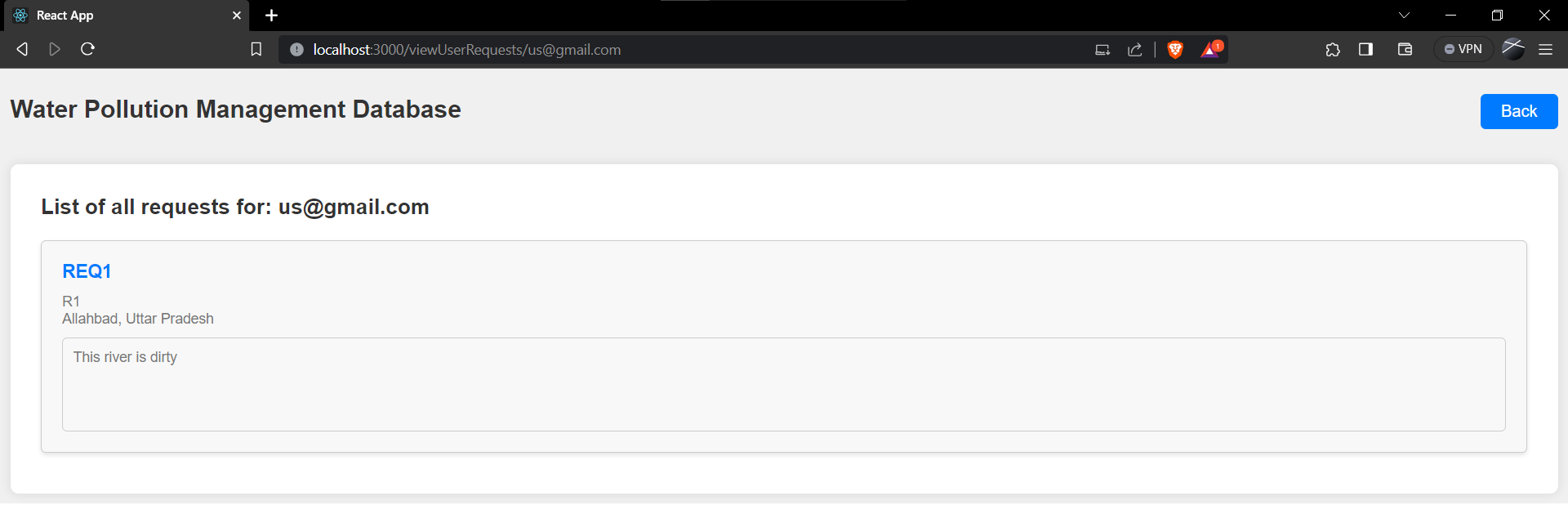


5) Raising public request regarding a specific water body

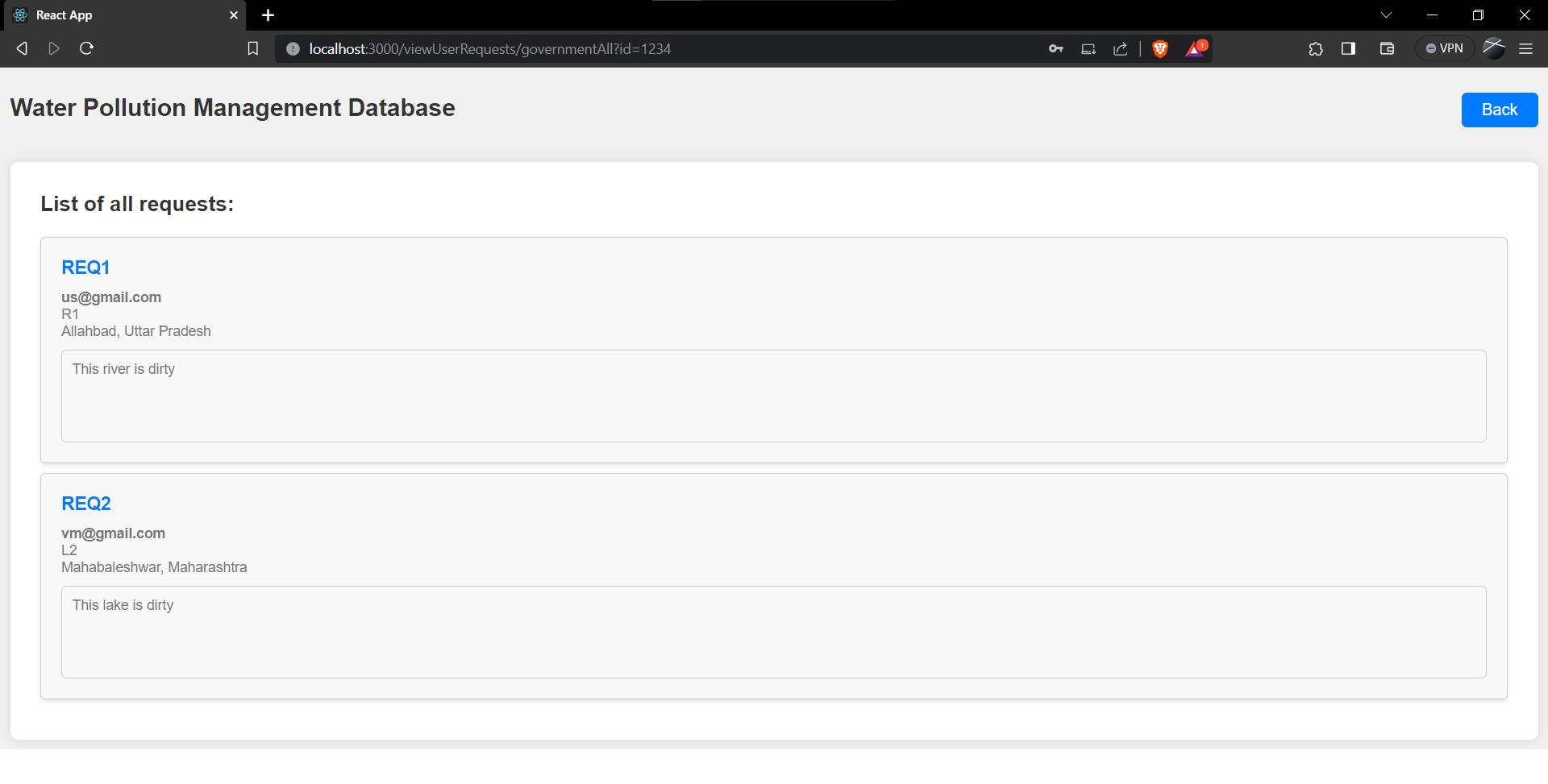
Adding Request:



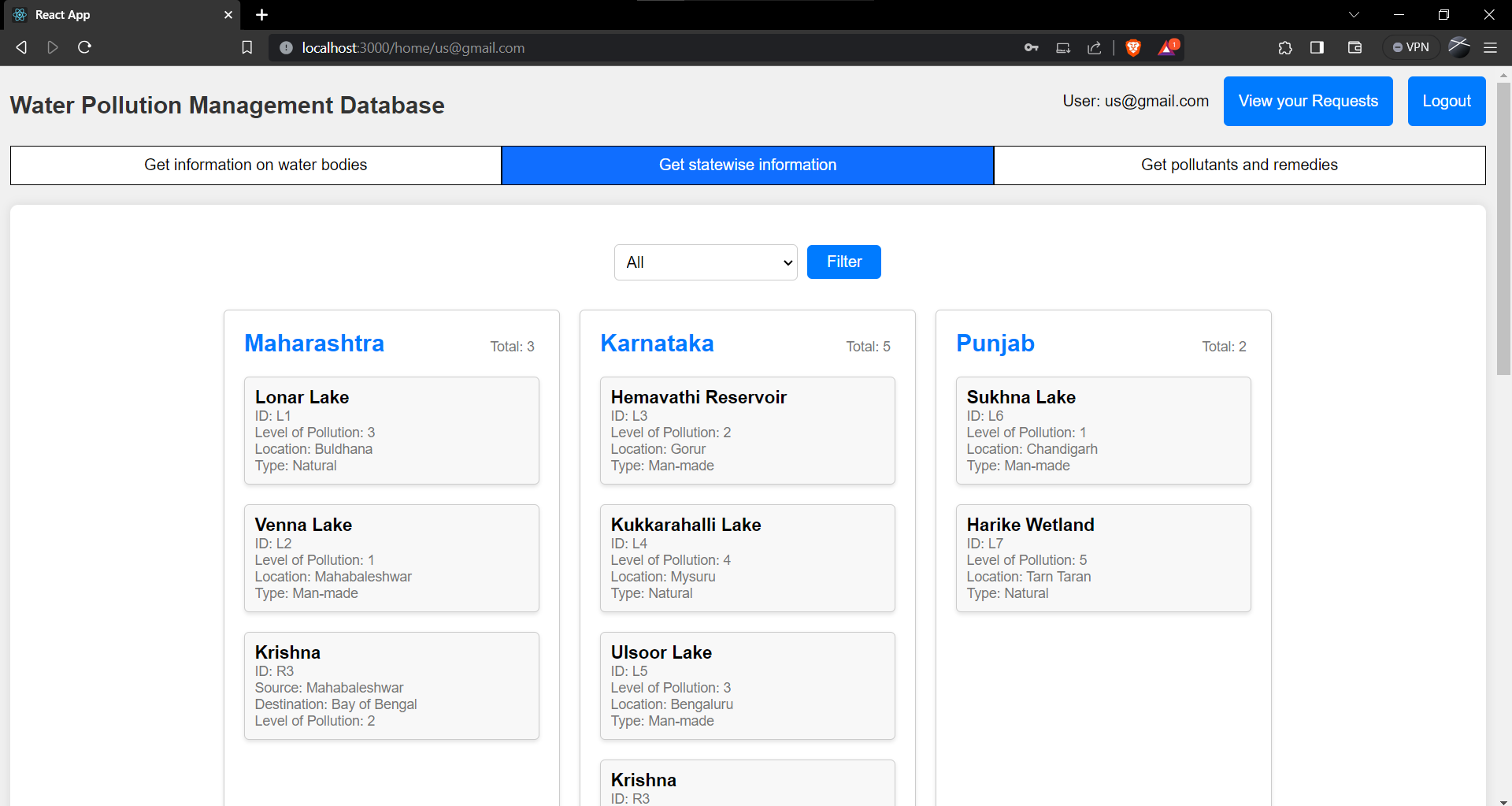
Viewing your requests as a user:

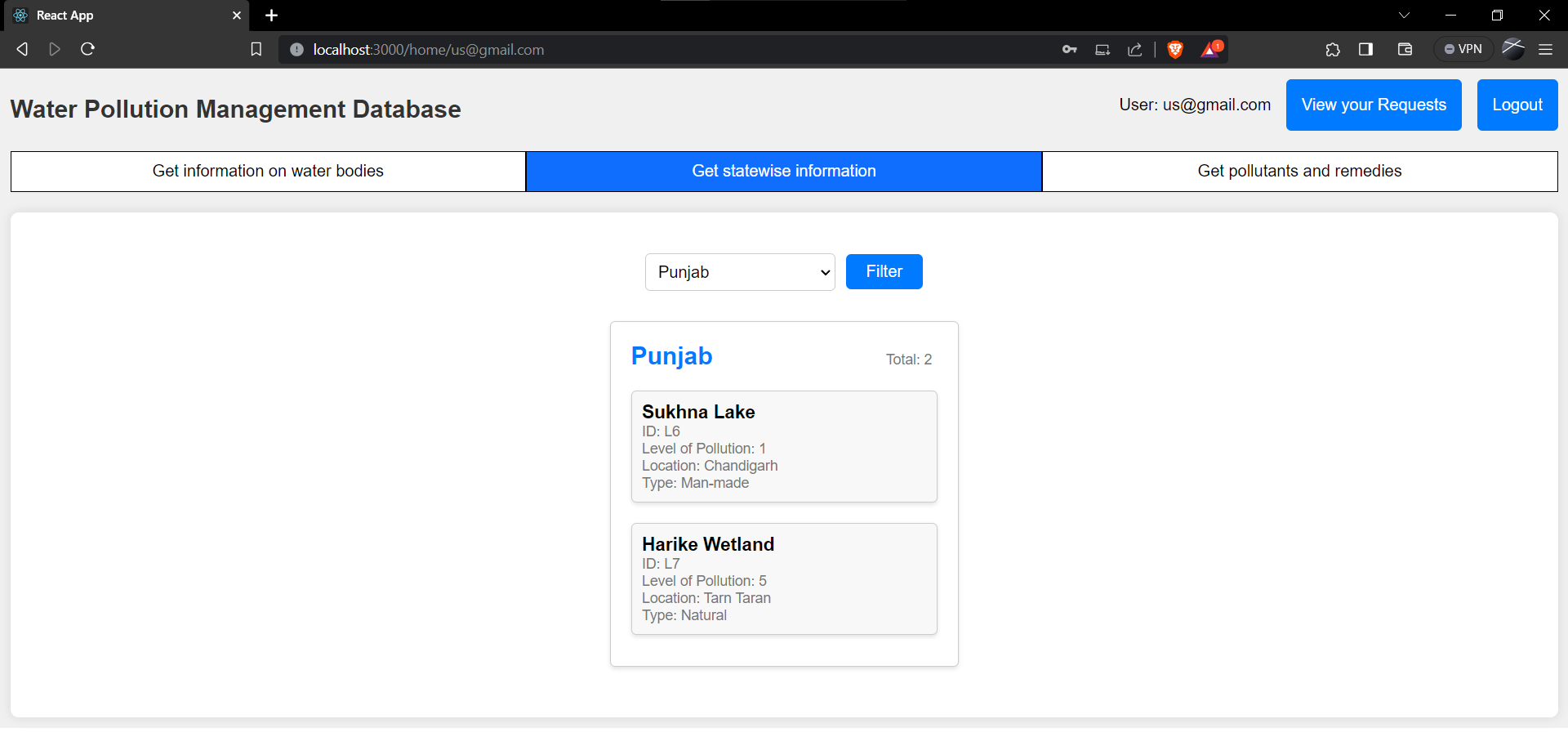


Viewing all requests as a government user:



6) Viewing state wise information





* **Procedures with Nested and Join queries, Functions with Aggregate queries and Triggers :**

1. **Procedures, Nested queries and Join queries**:

USE miniProject;

DELIMITER //

CREATE PROCEDURE GetLakeDetails(IN id VARCHAR(10))

BEGIN

WITH temp AS (

SELECT

L.lake\_id AS lake\_id,

L.name AS lake\_name,

L.level\_of\_pollution,

L.location,

L.type,

L.state\_name,

LP.pollutant\_id,

LP.concentration,

LP.effect,

P.name AS pollutant\_name,

P.chemical\_composition,

P.source,

P.type AS pollutant\_type,

P.severity

FROM

(SELECT \*

FROM lakes

WHERE lake\_id = id) AS L

JOIN lake\_pollutants AS LP ON L.lake\_id = LP.lake\_id

JOIN pollutants AS P ON LP.pollutant\_id = P.pollutant\_id

)

SELECT

temp.\*,

R.remedy\_id,

R.name AS remedy\_name,

R.effect AS remedy\_effect,

R.scale AS remedy\_scale,

R.avg\_price AS remedy\_avg\_price

FROM temp

JOIN remedies AS R ON temp.pollutant\_id = R.pollutant\_id;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE GetRiverDetails(IN id VARCHAR(10))

BEGIN

WITH temp AS (

SELECT

RI.river\_id AS river\_id,

RI.name AS river\_name,

RI.level\_of\_pollution,

RI.source AS source,

RI.destination AS destination,

RP.pollutant\_id,

RP.concentration,

RP.effect,

P.name AS pollutant\_name,

P.chemical\_composition,

P.source AS pollutant\_source,

P.type AS pollutant\_type,

P.severity

FROM

(SELECT \*

FROM rivers

WHERE river\_id = id) AS RI

JOIN river\_pollutants AS RP ON RI.river\_id = RP.river\_id

JOIN pollutants AS P ON RP.pollutant\_id = P.pollutant\_id

)

SELECT

temp.\*,

R.remedy\_id,

R.name AS remedy\_name,

R.effect AS remedy\_effect,

R.scale AS remedy\_scale,

R.avg\_price AS remedy\_avg\_price

FROM temp

JOIN remedies AS R ON temp.pollutant\_id = R.pollutant\_id;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE InsertUserDetails(IN user\_email VARCHAR(50))

BEGIN

SET @create\_user\_query = CONCAT("CREATE USER ", CONCAT\_WS('@', QUOTE(user\_email), 'localhost'), ";");

PREPARE create\_user\_stmt FROM @create\_user\_query;

EXECUTE create\_user\_stmt;

DEALLOCATE PREPARE create\_user\_stmt;

SET @grant\_query = CONCAT("GRANT 'users'@'localhost' TO ", CONCAT\_WS('@', QUOTE(user\_email), 'localhost'), ";");

PREPARE grant\_stmt FROM @grant\_query;

EXECUTE grant\_stmt;

DEALLOCATE PREPARE grant\_stmt;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE InsertGovernmentUserDetails(IN user\_email VARCHAR(50))

BEGIN

SET @create\_user\_query = CONCAT("CREATE USER ", CONCAT\_WS('@', QUOTE(user\_email), 'localhost'), ";");

PREPARE create\_user\_stmt FROM @create\_user\_query;

EXECUTE create\_user\_stmt;

DEALLOCATE PREPARE create\_user\_stmt;

SET @grant\_query = CONCAT("GRANT 'governmentUsers'@'localhost' TO ", CONCAT\_WS('@', QUOTE(user\_email), 'localhost'), ";");

PREPARE grant\_stmt FROM @grant\_query;

EXECUTE grant\_stmt;

DEALLOCATE PREPARE grant\_stmt;

END //

DELIMITER ;

DELIMITER //

CREATE PROCEDURE InsertAdminDetails(IN user\_email VARCHAR(50))

BEGIN

SET @create\_user\_query = CONCAT("CREATE USER ", CONCAT\_WS('@', QUOTE(user\_email), 'localhost'), ";");

PREPARE create\_user\_stmt FROM @create\_user\_query;

EXECUTE create\_user\_stmt;

DEALLOCATE PREPARE create\_user\_stmt;

SET @grant\_query = CONCAT("GRANT 'admins'@'localhost' TO ", CONCAT\_WS('@', QUOTE(user\_email), 'localhost'), ";");

PREPARE grant\_stmt FROM @grant\_query;

EXECUTE grant\_stmt;

DEALLOCATE PREPARE grant\_stmt;

END //

DELIMITER ;

1. **Functions, Aggregate queries :**

USE miniProject;

DELIMITER //

CREATE FUNCTION getTotalBodiesInState(input\_state VARCHAR(30))

RETURNS INT

DETERMINISTIC

BEGIN

DECLARE number\_of\_rivers INT;

DECLARE number\_of\_lakes INT;

SELECT COUNT(\*)

INTO number\_of\_rivers

FROM (

SELECT state\_name, river\_id

FROM rivers\_in\_state

WHERE state\_name = input\_state

) AS num\_rivers

GROUP BY state\_name;

SELECT COUNT(\*)

INTO number\_of\_lakes

FROM (

SELECT state\_name, lake\_id

FROM lakes

WHERE state\_name = input\_state

) AS num\_lakes

GROUP BY state\_name;

IF number\_of\_lakes IS NULL AND number\_of\_rivers IS NULL THEN

RETURN 0;

ELSEIF number\_of\_lakes IS NULL THEN

RETURN number\_of\_rivers;

ELSEIF number\_of\_rivers IS NULL THEN

RETURN number\_of\_lakes;

ELSE

RETURN number\_of\_rivers + number\_of\_lakes;

END IF;

END //

DELIMITER ;

1. **Trigger queries :**

USE miniProject;

DELIMITER //

CREATE TRIGGER InsertLake

BEFORE INSERT ON lakes

FOR EACH ROW

BEGIN

DECLARE number\_of\_rows INT;

SELECT COUNT(\*) INTO number\_of\_rows FROM lakes;

SET NEW.lake\_id = CONCAT("L", number\_of\_rows + 1);

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER InsertRiver

BEFORE INSERT ON rivers

FOR EACH ROW

BEGIN

DECLARE number\_of\_rows INT;

SELECT COUNT(\*) INTO number\_of\_rows FROM rivers;

SET NEW.river\_id = CONCAT("R", number\_of\_rows + 1);

END //

DELIMITER ;

DELIMITER //

CREATE TRIGGER InsertRequest

BEFORE INSERT ON requests

FOR EACH ROW

BEGIN

DECLARE number\_of\_rows INT;

SELECT COUNT(\*) INTO number\_of\_rows FROM requests;

SET NEW.request\_id = CONCAT("REQ", number\_of\_rows + 1);

END //

DELIMITER ;

* **Code snippets for invoking the Procedures/Functions/Trigger**

**1) Procedures:**

database.query(query, [...values, request.body.governmentID], (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

var userQuery

if(request.body.governmentID == null) {

userQuery = "**CALL InsertUserDetails(?);**"

} else {

userQuery = "**CALL InsertGovernmentUserDetails(?);**"

}

database.query(userQuery, request.body.email, (error, userResult) => {

if(error) {

console.log(error)

return response.json(error)

}

return response.json(result)

})

})

app.get("/viewLakeDetails/:lakeID", (request, response) => {

const query = "**CALL GetLakeDetails(?);**"

database.query(query, request.params.lakeID, (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

const data = result[0]

if(data.length === 0) {

return response.json([])

}

const lakeInfo = {

lake\_id: data[0].lake\_id,

lake\_name: data[0].lake\_name,

level\_of\_pollution: data[0].level\_of\_pollution,

location: data[0].location,

type: data[0].type,

state\_name: data[0].state\_name

}

const pollutants = {}

for(var i in data) {

if(!pollutants[data[i]["pollutant\_id"]]) {

pollutants[data[i]["pollutant\_id"]] = {

pollutant\_id: data[i]["pollutant\_id"],

pollutant\_name: data[i]["pollutant\_name"],

concentration: data[i]["concentration"],

effect: data[i]["effect"],

chemical\_composition: data[i]["chemical\_composition"],

source: data[i]["source"],

pollutant\_type: data[i]["pollutant\_type"],

severity: data[i]["severity"],

remedies: {}

}

}

if(!pollutants[data[i]["pollutant\_id"]]["remedies"]["remedy\_id"]) {

pollutants[data[i]["pollutant\_id"]]["remedies"][data[i]["remedy\_id"]] = {

remedy\_id: data[i]["remedy\_id"],

remedy\_name: data[i]["remedy\_name"],

remedy\_scale: data[i]["remedy\_scale"],

remedy\_effect: data[i]["remedy\_effect"],

avg\_price : data[i]["remedy\_avg\_price"]

}

}

}

return response.json([

lakeInfo,

pollutants

])

})

})

app.get("/viewRiverDetails/:riverID", (request, response) => {

const query = "**CALL GetRiverDetails(?);**"

database.query(query, request.params.riverID, (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

const data = result[0]

if(data.length === 0) {

return response.json([])

}

const lakeInfo = {

river\_id: data[0].river\_id,

river\_name: data[0].river\_name,

level\_of\_pollution: data[0].level\_of\_pollution,

source: data[0].source,

destination: data[0].destination

}

const pollutants = {}

for(var i in data) {

if(!pollutants[data[i]["pollutant\_id"]]) {

pollutants[data[i]["pollutant\_id"]] = {

pollutant\_id: data[i]["pollutant\_id"],

pollutant\_name: data[i]["pollutant\_name"],

concentration: data[i]["concentration"],

effect: data[i]["effect"],

chemical\_composition: data[i]["chemical\_composition"],

pollutant\_source: data[i]["pollutant\_source"],

pollutant\_type: data[i]["pollutant\_type"],

severity: data[i]["severity"],

remedies: {}

}

}

if(!pollutants[data[i]["pollutant\_id"]]["remedies"]["remedy\_id"]) {

pollutants[data[i]["pollutant\_id"]]["remedies"][data[i]["remedy\_id"]] = {

remedy\_id: data[i]["remedy\_id"],

remedy\_name: data[i]["remedy\_name"],

remedy\_scale: data[i]["remedy\_scale"],

remedy\_effect: data[i]["remedy\_effect"],

avg\_price : data[i]["remedy\_avg\_price"]

}

}

}

return response.json([

lakeInfo,

pollutants

])

})

})

**2) Functions**

app.get("/getTotalBodiesInState", (request, response) => {

const query = "SELECT name, **getTotalBodiesInState(name)** AS total\_count from states;"

database.query(query, (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

const counts = {}

for(var i in result) {

counts[result[i]["name"]] = result[i]["total\_count"]

}

return response.json(counts)

})

})

**3) Triggers**

app.post("/insertLake", (request, response) => {

const query = "INSERT INTO lakes VALUES ('', ?, ?, ?, ?, ?);"

const values = [

request.body.name,

request.body.level,

request.body.location,

request.body.type,

request.body.state

]

database.query(query, [...values], (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

return response.json(result)

})

})

app.post("/insertRiver", (request, response) => {

const query = "INSERT INTO rivers VALUES ('', ?, ?, ?, ?);"

const values = [

request.body.name,

request.body.level,

request.body.source,

request.body.destination

]

database.query(query, [...values], (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

return response.json(result)

})

})

app.post("/newRequest", (request, response) => {

var query = ""

if(request.body.body\_id[0] === "R") {

query = "INSERT INTO requests (`request\_id`, `user\_email`, `river\_id`, `city`, `state\_name`, `content`) VALUES ('', ?, ?, ?, ?, ?);"

} else {

query = "INSERT INTO requests (`request\_id`, `user\_email`, `lake\_id`, `city`, `state\_name`, `content`) VALUES ('', ?, ?, ?, ?, ?);"

}

const values = [

request.body.user\_email,

request.body.body\_id,

request.body.city,

request.body.state,

request.body.content

]

database.query(query, [...values], (error, result) => {

if(error) {

console.log(error)

return response.json(error)

}

return response.json(result)

})

})

**The triggers are called before inserting into the lakes, rivers and requests tables as shown above.**