Name: Sarah Kimberly Fischer
Student ID: 001202300177
Class: IT-2023 Class 1
Subject: Distributed and Parallel System

Project Overview

This project is a full-stack personal expense tracker web application. It allows users to record and categorize their expenses. The frontend is built with React (Vite) and the backend with Node.js (Express), while the data is stored in a PostgreSQL database. Everything is containerized using Docker, ensuring consistent environments and simplified deployment.

Tech Stack

- **Frontend**: React (Vite)
- **Backend**: Node.js (Express), Sequelize
- **Database**: PostgreSQL
- **Deployment**: Docker, Docker Compose
- **Reverse Proxy**: Nginx

System Architecture

- **frontend/**: React app served by Nginx
- **backend/**: Express API for CRUD operations
- **postgres**: PostgreSQL container for persistent data
- **docker-compose.yml**: coordinates all services

Features

- **Add Expenses**: Input amount, description, category, and date.
- **View Expenses**: List all transactions with filters by date and category.
- **Edit/Delete**: Modify or remove existing expense records.

- **Categories**: Choose from predefined options like Food, Transport, Entertainment, etc.
- **Summary Dashboard**: View total spending and category breakdown.
- **Pagination**: Navigate expense records with page and limit query support.
- **API Routing via Nginx**: All API requests are routed through a reverse proxy.
- **PostgreSQL with Sequelize**: Structured relational storage and ORM integration.
- **Fully Containerized**: Deployed using Docker and managed with Docker Compose.

Folder Structure

project-root/

├── backend/

│ ├── Dockerfile

│ ├── server.js

│ └── ...

├── frontend/

│ ├── Dockerfile

│ ├── nginx.conf

│ └── ...

├── docker-compose.yml

├── .env

├── .env.example

└── README.md

Step-by-Step: Deploying the App with Docker

1. **Clone the repository**

```
git clone https://github.com/sarahkimberlyy/expense-tracker.git
cd expense-tracker
```

2. **Create `.env` file from template**

```
cp .env.example .env
```

Copy the example environment file and edit it with your database credentials

Make sure .env contains:

```
POSTGRES_DB=expenses

POSTGRES_USER=postgres

POSTGRES_PASSWORD=yourpassword
```

3. **Run Docker Compose**

```
docker-compose up --build
```

This command will:

- Build the backend and frontend images
- Set up a PostgreSQL database
- Start all services in a shared network

4. **Access the App**

Frontend: http://localhost:3000

Backend (direct): http://localhost:3001/api/expenses (API endpoints served by Express)

Frontend → Backend: http://localhost:3000/api/expenses (API requests proxied via frontend - reverse proxy setup)

Docker Breakdown

1. **Backend Dockerfile**

```
# Use lightweight Node.js 20 base image
FROM node:20-alpine
```

```
# Set working directory inside the container
WORKDIR /app
# Copy package.json and package-lock.json
COPY package*.json ./
# Install backend dependencies
RUN npm install
# Copy the rest of the application code
COPY . .
# Expose backend port for API
EXPOSE 3001
# Start the backend server
CMD ["npm", "start"]
```

Summary:

- Sets up the Node.js backend server using a lightweight `node:20-alpine` image.
- Installs dependencies and runs the Express server.
- Exposes port `3001` for API access.
- Used by Docker Compose to containerize the backend.

2. Frontend Dockerfile

```
# --- Build Stage ---
# Use lightweight Node.js image to build the React (Vite) frontend
FROM node:20-alpine AS builder
# Set working directory inside the container
WORKDIR /app
# Copy dependency files first for caching purposes
COPY package*.json ./
# Install frontend dependencies
RUN npm install
# Copy the rest of the application source code
COPY . .
# Build the production-ready frontend (output goes to /app/dist)
RUN npm run build
# --- Serve Stage ---
# Use Nginx to serve the built static files
```

```
FROM nginx:alpine
# Copy built files from previous stage to Nginx's default public directory
COPY --from=builder /app/dist /usr/share/nginx/html
# Replace default Nginx config with custom config for routing and API
proxying
COPY nginx.conf /etc/nginx/conf.d/default.conf
# Expose port 80 to allow web traffic
EXPOSE 80
# Run Nginx in the foreground
CMD ["nginx", "-g", "daemon off;"]
```

Summary:

- Multi-stage build:
    - Stage 1: Builds React app with `npm run build` using `node:20-alpine`.
    - Stage 2: Serves the built static files using `nginx:alpine`.
- Copies a custom `nginx.conf` for SPA routing and API proxy.
- Exposes port `80` to serve the frontend.

3. Nginx Config (nginx.conf)

```
server {
  listen 80;  # Listen on port 80 for incoming HTTP requests
  location / {
    root /usr/share/nginx/html;  # Serve static files from this directory
(frontend build)
    index index.html;            # Default file to serve
    try_files $uri /index.html;  # Redirect all unmatched routes to
index.html (for SPA routing)
  }
  location /api/ {
    proxy_pass http://backend:3001;  # Forward /api requests to backend
container on port 3001
    proxy_http_version 1.1;          # Use HTTP/1.1 for compatibility
    proxy_set_header Host $host;     # Pass original host header
```

```
    proxy_set_header X-Real-IP $remote_addr;                # Forward real
client IP
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;  #
Forward all proxy IPs
    proxy_set_header X-Forwarded-Proto $scheme;             # Forward
original protocol (http or https)
  }
}
```

Summary:

- Serves the React frontend from /usr/share/nginx/html.
- Uses try_files $uri /index.html to support client-side routing.
- Proxies all /api/ requests to the backend at http://backend:3001.

4. **docker-compose.yml**

```
services:
  frontend:
    build:
      context: ./frontend          # Build the frontend image from the
./frontend directory
    ports:
      - "3000:80"                  # Map container port 80 (used by nginx
or similar in frontend) to host port 3000
    depends_on:
      - backend                   # Wait for backend to be ready before
starting frontend
    networks:
      - app-network               # Connect to a custom bridge network for
inter-service communication

  backend:
    build:
      context: ./backend           # Build the backend image from the
./backend directory
    ports:
```

```
      - "3001:3001"                # Map container port 3001 to host port
3001
    depends_on:
      - postgres                   # Ensure Postgres is ready before
starting backend
    environment:
      -
DATABASE_URL=postgres://${POSTGRES_USER}:${POSTGRES_PASSWORD}@postgres:543
2/${POSTGRES_DB}
                                   # Environment variable for database
connection string (uses service name 'postgres' as hostname)
    networks:
      - app-network                # Connect backend to the same network as
frontend and database

  postgres:
    image: postgres:15             # Use the official Postgres 15 image
    restart: always                # Automatically restart the container if
it crashes
    environment:
      - POSTGRES_DB=${POSTGRES_DB}             # Set initial database name
from .env
      - POSTGRES_USER=${POSTGRES_USER}         # Set Postgres user from
.env
      - POSTGRES_PASSWORD=${POSTGRES_PASSWORD} # Set Postgres password
from .env
    volumes:
      - pgdata:/var/lib/postgresql/data        # Persist database data
using a named volume
    ports:
      - "5432:5432"                # Map container port 5432 to host for
development access (e.g., via pgAdmin)
    networks:
      - app-network                # Connect Postgres to the custom bridge
network

volumes:
  pgdata:                          # Declare a named volume for persisting
Postgres data
```

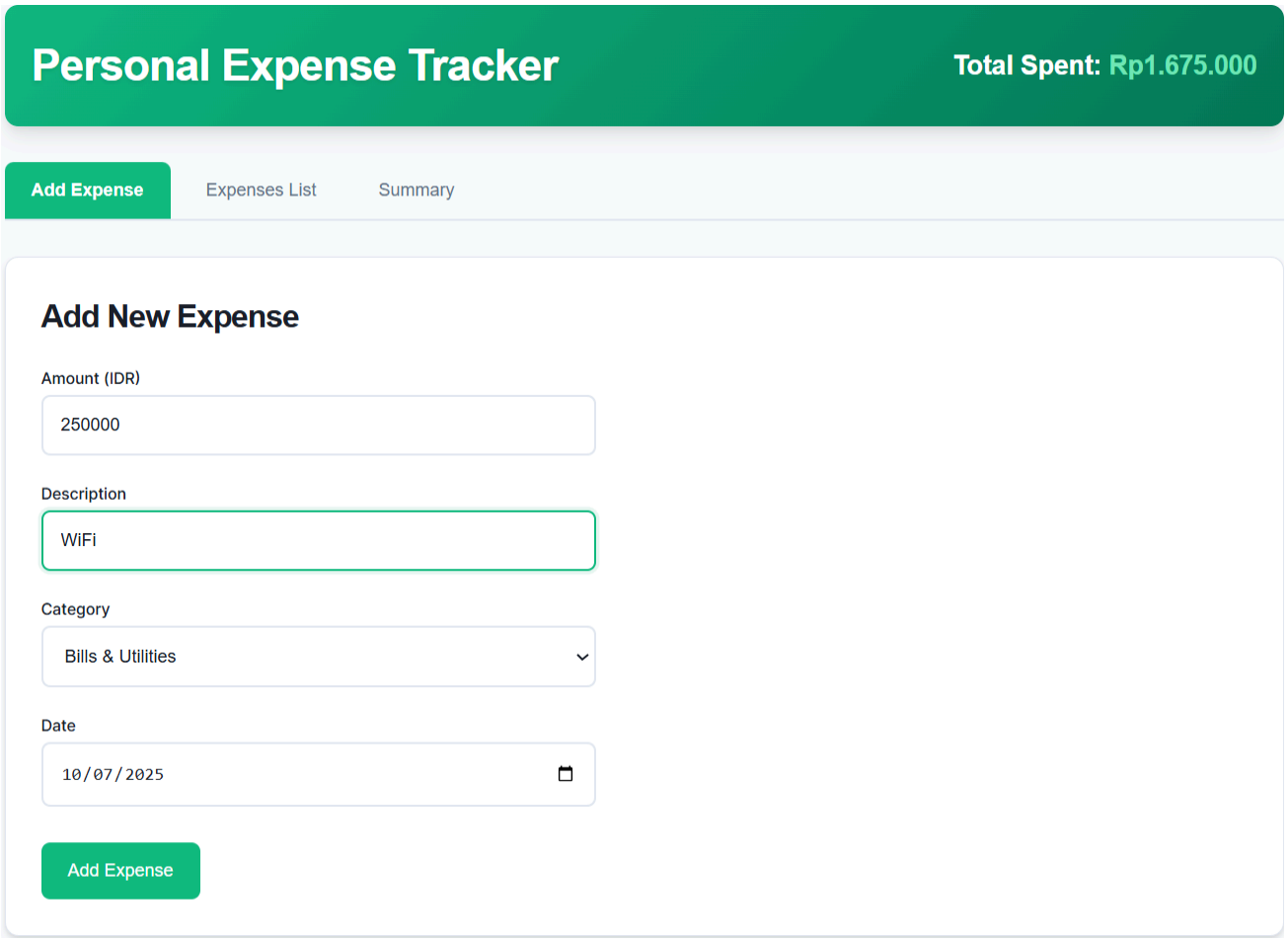
```yaml
networks:
  app-network:                    # Define a custom bridge network for
service communication
    driver: bridge
```

Summary:

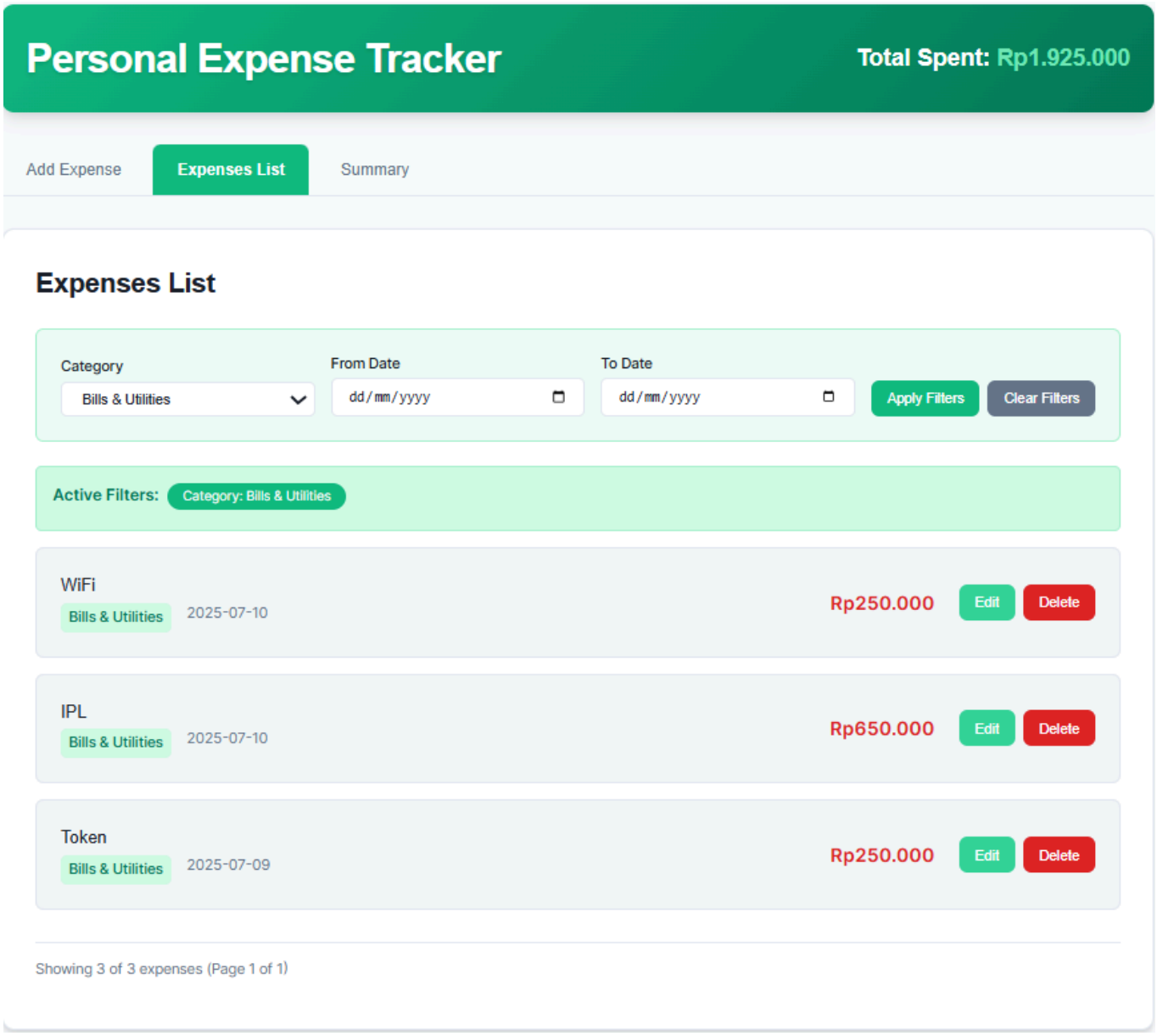
- Three services: backend, frontend, and postgres
- Uses Dockerfiles for frontend and backend
- PostgreSQL is connected via environment variables
- Services are connected through a shared custom network
- Frontend runs on localhost:3000, Backend on localhost:3001
- Data is persisted in a named volume pgdata

Screenshots

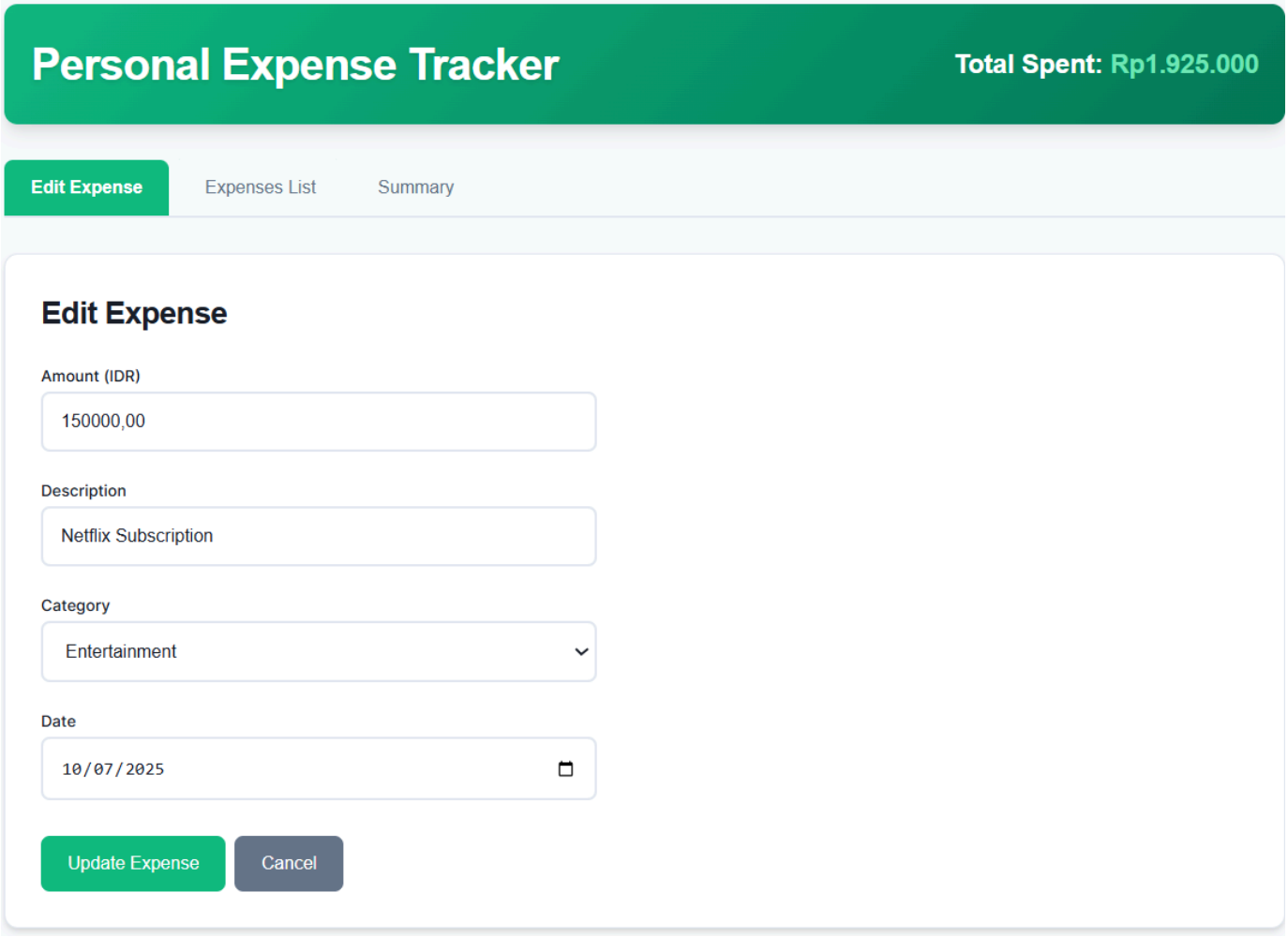
1. Add New Expense



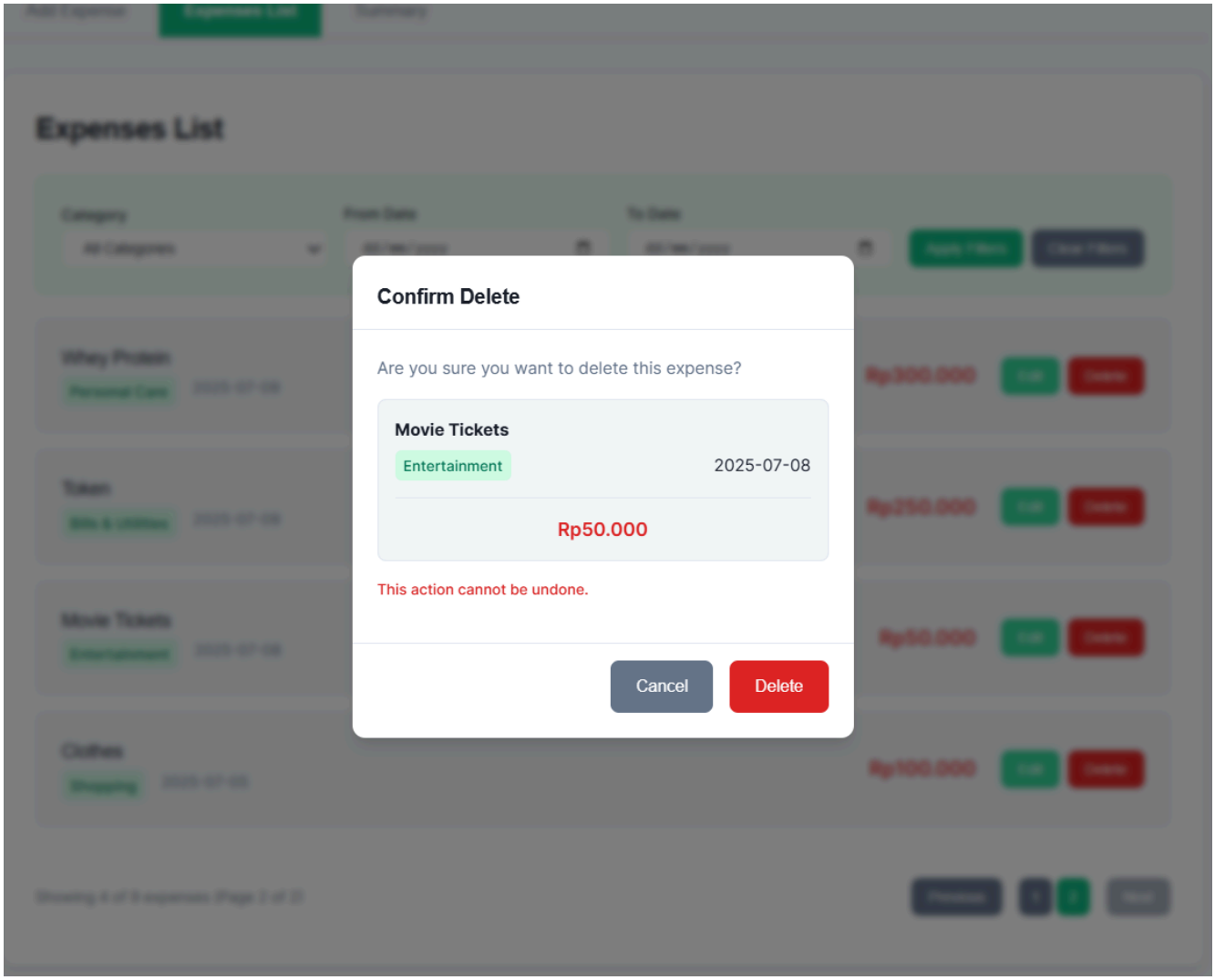
2. View Expenses List (Filters Available)



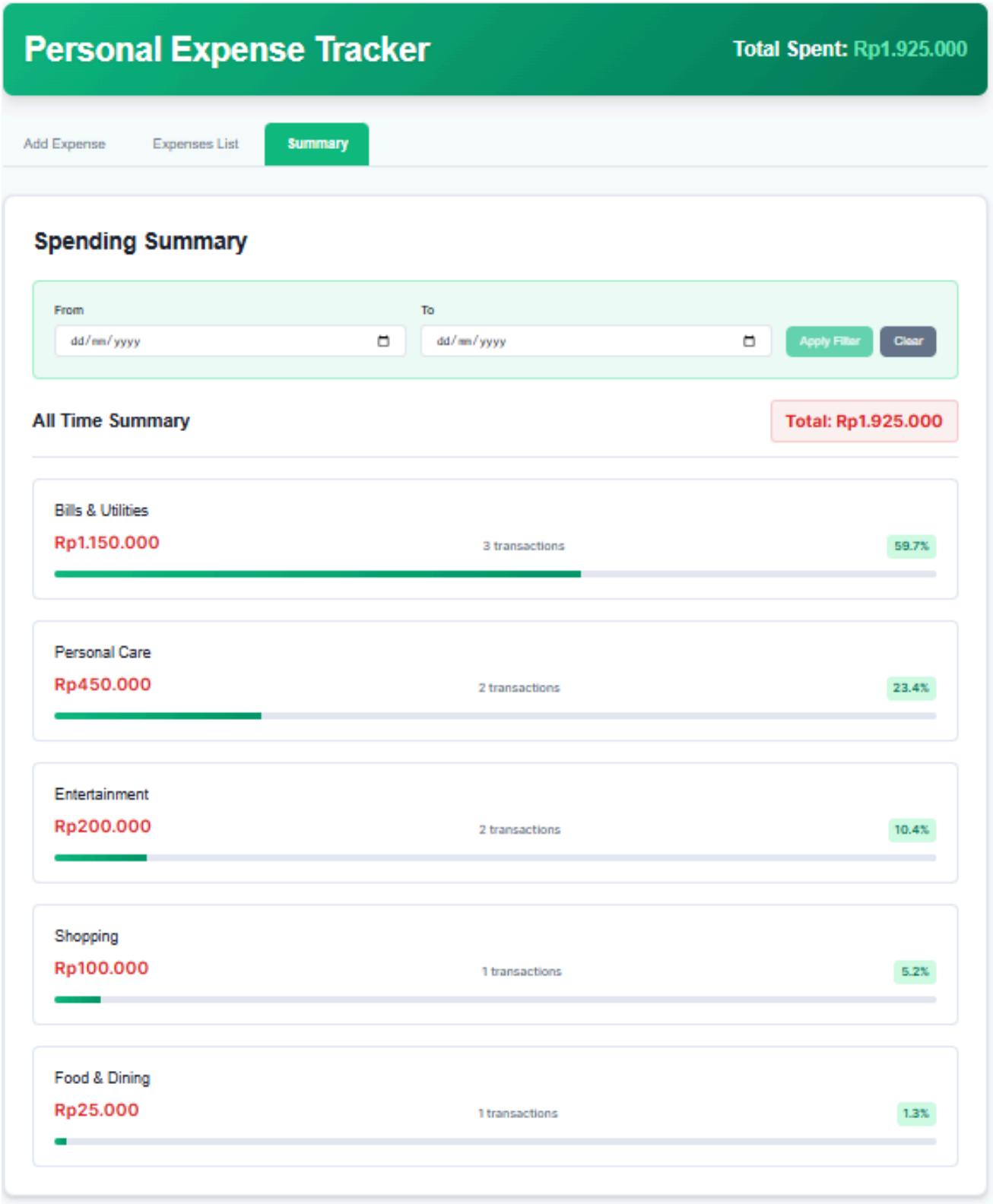
3. Edit Expenses



4. Delete Expense



5. **Summary Dashboard (Filters Available)**



Github Link

https://github.com/sarahkimberlyy/expense-tracker