Expense Management API

1. Project Overview

Build a REST API for managing personal expenses.

The API allows users to:

- Add expenses
- View all expenses
- View one expense
- Update expenses
- Delete expenses
- Search expenses
- Calculate total expenses
- Filter expenses by category

FastAPI uses Pydantic models for request validation and response models for structured API responses.

2. Features

- REST API
- Create, read, update and delete expenses
- SQLite database
- Category filtering
- Expense search
- Total expense calculation
- Request validation
- Automatic Swagger documentation
- Parameterized SQL queries

3. Technologies

- Python 3
- FastAPI
- Pydantic
- SQLite
- Uvicorn
- sqlite3

SQLite parameterized queries are used instead of constructing SQL with user input directly.

4. Folder Structure

```
expense_management_api/
□□□ main.py
□□□ database.py
□□□ schemas.py
□□□ requirements.txt
□□□ README.md
```

5. How the Project Works

```
Client
  □
FastAPI
  □
Pydantic Validation
  □
API Endpoint
  □
SQLite Database
  □
JSON Response
```

Example:

POST /expenses

Validate expense

Save to SQLite

Return created expense

6. Complete Backend Code

database.py

```
import sqlite3


class ExpenseDatabase:

    def __init__(self, database_name="expenses.db"):
        self.database_name = database_name
        self.create_table()

    def connect(self):
        return sqlite3.connect(self.database_name)

    def create_table(self):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            CREATE TABLE IF NOT EXISTS expenses (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                title TEXT NOT NULL,
                amount REAL NOT NULL,
```

```
            category TEXT NOT NULL,
            date TEXT NOT NULL
        )
    """)

    connection.commit()
    connection.close()

def create_expense(self, title, amount, category, date):
    connection = self.connect()
    cursor = connection.cursor()

    cursor.execute("""
        INSERT INTO expenses
        (title, amount, category, date)
        VALUES (?, ?, ?, ?)
    """, (title, amount, category, date))

    connection.commit()
    expense_id = cursor.lastrowid
    connection.close()

    return expense_id

def get_expenses(self):
    connection = self.connect()
    connection.row_factory = sqlite3.Row
    cursor = connection.cursor()

    cursor.execute("""
        SELECT id, title, amount, category, date
        FROM expenses
        ORDER BY id DESC
    """)
```

```
        expenses = [dict(row) for row in cursor.fetchall()]
        connection.close()

        return expenses

    def get_expense(self, expense_id):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, title, amount, category, date
            FROM expenses
            WHERE id = ?
        """, (expense_id,))

        expense = cursor.fetchone()
        connection.close()

        return dict(expense) if expense else None

    def update_expense(
        self,
        expense_id,
        title,
        amount,
        category,
        date
    ):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
```

```
            UPDATE expenses
            SET title = ?,
                amount = ?,
                category = ?,
                date = ?
            WHERE id = ?
        """, (
            title,
            amount,
            category,
            date,
            expense_id
        ))

        connection.commit()
        updated = cursor.rowcount > 0
        connection.close()

        return updated

    def delete_expense(self, expense_id):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            DELETE FROM expenses
            WHERE id = ?
        """, (expense_id,))

        connection.commit()
        deleted = cursor.rowcount > 0
        connection.close()

        return deleted
```

```
    def search_expenses(self, keyword):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        pattern = f"%{keyword}%"

        cursor.execute("""
            SELECT id, title, amount, category, date
            FROM expenses
            WHERE title LIKE ?
               OR category LIKE ?
            ORDER BY id DESC
        """, (pattern, pattern))

        expenses = [dict(row) for row in cursor.fetchall()]
        connection.close()

        return expenses

    def get_by_category(self, category):
        connection = self.connect()
        connection.row_factory = sqlite3.Row
        cursor = connection.cursor()

        cursor.execute("""
            SELECT id, title, amount, category, date
            FROM expenses
            WHERE category = ?
            ORDER BY id DESC
        """, (category,))

        expenses = [dict(row) for row in cursor.fetchall()]
```

```
        connection.close()

        return expenses

    def get_total(self):
        connection = self.connect()
        cursor = connection.cursor()

        cursor.execute("""
            SELECT COALESCE(SUM(amount), 0)
            FROM expenses
        """)

        total = cursor.fetchone()[0]
        connection.close()

        return total
```

schemas.py

```
from pydantic import BaseModel, Field


class ExpenseCreate(BaseModel):
    title: str = Field(min_length=1, max_length=200)
    amount: float = Field(gt=0)
    category: str = Field(min_length=1, max_length=100)
    date: str = Field(min_length=1, max_length=30)


class Expense(ExpenseCreate):
    id: int
```

main.py

```
from fastapi import FastAPI, HTTPException

from database import ExpenseDatabase
from schemas import Expense, ExpenseCreate


app = FastAPI(
    title="Expense Management API",
    description="A simple REST API for managing expenses.",
    version="1.0.0"
)

database = ExpenseDatabase()


@app.get("/")
def home():
    return {
        "message": "Expense Management API is running"
    }


@app.post(
    "/expenses",
    response_model=Expense,
    status_code=201
)
def create_expense(expense: ExpenseCreate):
    expense_id = database.create_expense(
        expense.title,
        expense.amount,
        expense.category,
        expense.date
    )
```

```
    return {
        "id": expense_id,
        **expense.model_dump()
    }


@app.get(
    "/expenses",
    response_model=list[Expense]
)
def get_expenses():
    return database.get_expenses()


@app.get(
    "/expenses/{expense_id}",
    response_model=Expense
)
def get_expense(expense_id: int):
    expense = database.get_expense(expense_id)

    if expense is None:
        raise HTTPException(
            status_code=404,
            detail="Expense not found."
        )

    return expense


@app.put(
    "/expenses/{expense_id}",
    response_model=Expense
```

```
)
def update_expense(
    expense_id: int,
    expense: ExpenseCreate
):
    updated = database.update_expense(
        expense_id,
        expense.title,
        expense.amount,
        expense.category,
        expense.date
    )

    if not updated:
        raise HTTPException(
            status_code=404,
            detail="Expense not found."
        )

    return {
        "id": expense_id,
        **expense.model_dump()
    }


@app.delete("/expenses/{expense_id}")
def delete_expense(expense_id: int):
    deleted = database.delete_expense(expense_id)

    if not deleted:
        raise HTTPException(
            status_code=404,
            detail="Expense not found."
        )
```

```
    return {
        "message": "Expense deleted successfully."
    }


@app.get(
    "/expenses/search/{keyword}",
    response_model=list[Expense]
)
def search_expenses(keyword: str):
    return database.search_expenses(keyword)


@app.get(
    "/expenses/category/{category}",
    response_model=list[Expense]
)
def expenses_by_category(category: str):
    return database.get_by_category(category)


@app.get("/expenses-summary")
def expense_summary():
    return {
        "total_expenses": database.get_total()
    }
```

requirements.txt

```
fastapi
uvicorn
```

README.md

Expense Management API

A REST API for managing personal expenses.

Setup

python -m venv venv

Activate the virtual environment.

Windows:
venv\Scripts\activate

Install dependencies:

pip install -r requirements.txt

Run the API:

uvicorn main:app --reload

Open:

http://127.0.0.1:8000/docs

7. API Endpoints

POST /expenses
GET /expenses
GET /expenses/{expense_id}
PUT /expenses/{expense_id}
DELETE /expenses/{expense_id}

GET /expenses/search/{keyword}
GET /expenses/category/{category}

```
GET    /expenses-summary
```

FastAPI automatically generates interactive API documentation at `/docs`.

8. Example Request

Create Expense

```
POST /expenses
```

```
{
  "title": "Grocery Shopping",
  "amount": 4500,
  "category": "Food",
  "date": "2026-09-25"
}
```

Response:

```
{
  "id": 1,
  "title": "Grocery Shopping",
  "amount": 4500,
  "category": "Food",
  "date": "2026-09-25"
}
```

Search

```
GET /expenses/search/grocery
```

Category Filter

```
GET /expenses/category/Food
```

Total

`GET /expenses-summary`

Example response:

```
{
  "total_expenses": 12500
}
```

9. Basic Testing

Test these operations through Swagger:

```
1. Create an expense
2. Get all expenses
3. Get one expense by ID
4. Search an expense
5. Filter by category
6. Check total expenses
7. Update an expense
8. Delete an expense
```

Validation example:

```
{
  "title": "Invalid Expense",
  "amount": -500,
  "category": "Food",
  "date": "2026-09-25"
}
```

The API rejects the request because the amount must be greater than zero.

10. Small Improvements

Possible next improvements:

- User authentication
- Monthly expense summaries
- Income tracking
- Budget limits
- Pagination
- Date range filtering
- Category totals
- JWT authentication
- PostgreSQL
- React frontend
- Expense charts
- Deployment

Visit haas.dev for more resources and guides.