DBMS Project Presentation

Database Management System

Team Name : Team Trio

Team Code : K

Section : CSE-DS-4A

Course Code : 23DS2403

Project Title : Budget Buddy-(Money Management)

Team Members :

Kammala Kalyan	ENG23DS0063
Kovouru Venkata Naga Sai Pranav	ENG23DS0067
Amarnath Gowda KM	ENG23DS0052

Introduction

- **What is Budget Buddy?**
  A simple web app that helps you plan and track your personal monthly budget.
- **Objective of the Project**
  To give anyone—no matter their experience level—a clear way to set a budget, record expenses, and see if they stayed under budget each month.
- **Scope of the Project**
  - Users can register and log in.
  - Create a new monthly budget with categories (e.g. groceries, bills).
  - Add expenses against those categories.
  - View progress bars and charts showing planned vs. actual spending.
  - Track a “streak” of months under budget for extra motivation.

Budget Buddy

A Personal Monthly Budget Tracker

Problem Statement

- **Description of the problem being addressed**
  Many people struggle to keep track of where their money goes each month. Without a clear way to set limits and record expenses, it's easy to overspend, miss bills, and lose sight of financial goals.

- **Importance of the problem for this project**
  - **Financial Stress Reduction:** Helping users see their spending at a glance minimizes surprises and anxiety.
  - **Better Decision-Making:** With clear budget data, users can adjust habits, cut unnecessary costs, and save more effectively.
  - **Accessibility:** Beginners need a simple, guided tool—without it, they may never establish good budgeting habits.

System Architecture

System Architecture of Your Budget Buddy Project

1.Client (Browser)

- Renders HTML/CSS from Jinja2 templates.
- Uses JavaScript to fetch category data (/get-categories) and submits forms.

2.Web/Application Server

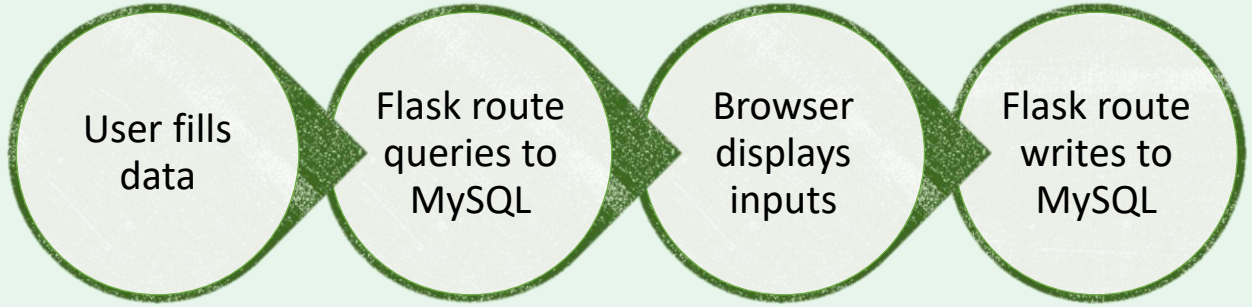
- **Flask's built-in WSGI server** (via app.run(debug=True)) handles all incoming HTTP(S) requests.
- Routes implement:
  - **Authentication & Sessions** (/login, /register, login_required)
  - **Budget Management** (/budget-setup, streak logic)
  - **Expense Logging** (/add-expense, JSON API /get-categories)
  - **Reporting & History** (/budget-progress, /reports, /history, /monthly-reset, /reset-all)

3.Data Access Layer

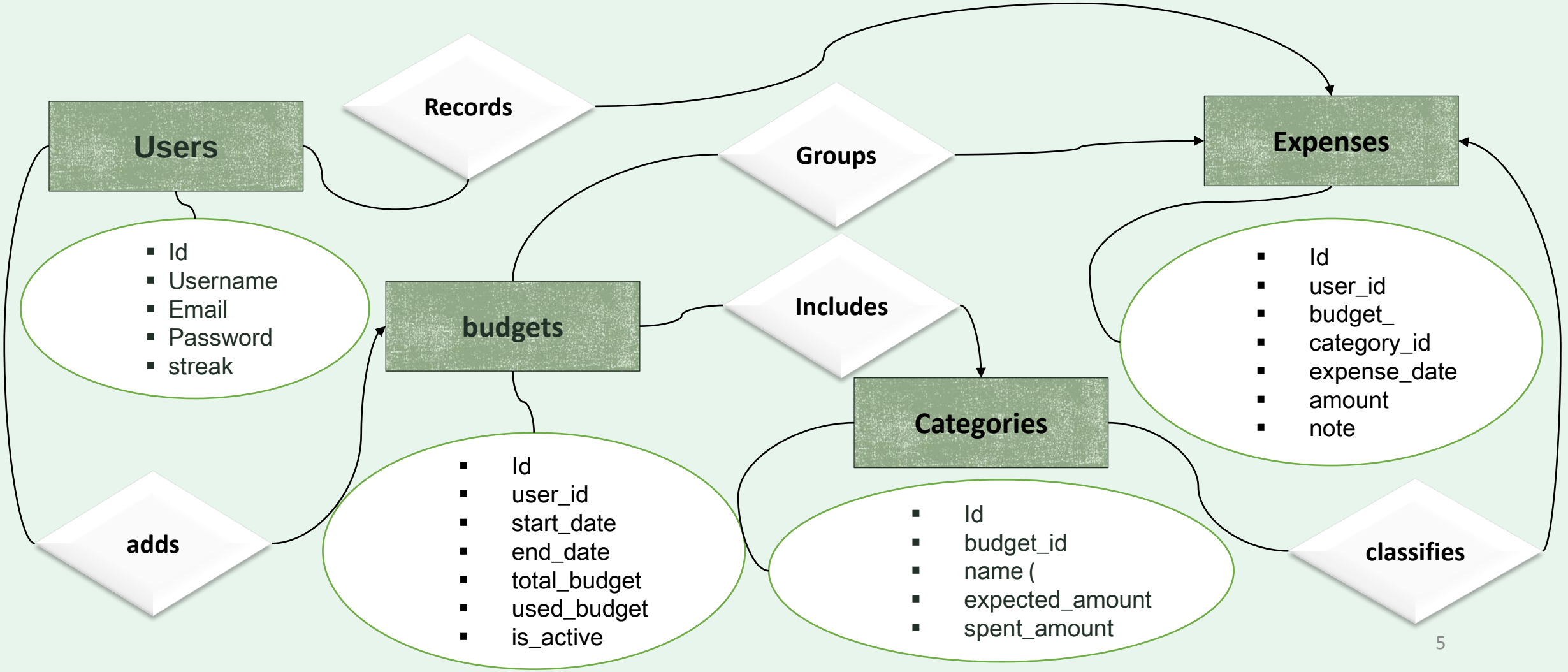
- **flask_mysqldb** extension connects Flask to MySQL.
- Raw SQL queries against four tables:
  - users
  - budgets
  - categories
  - expenses

4.Database

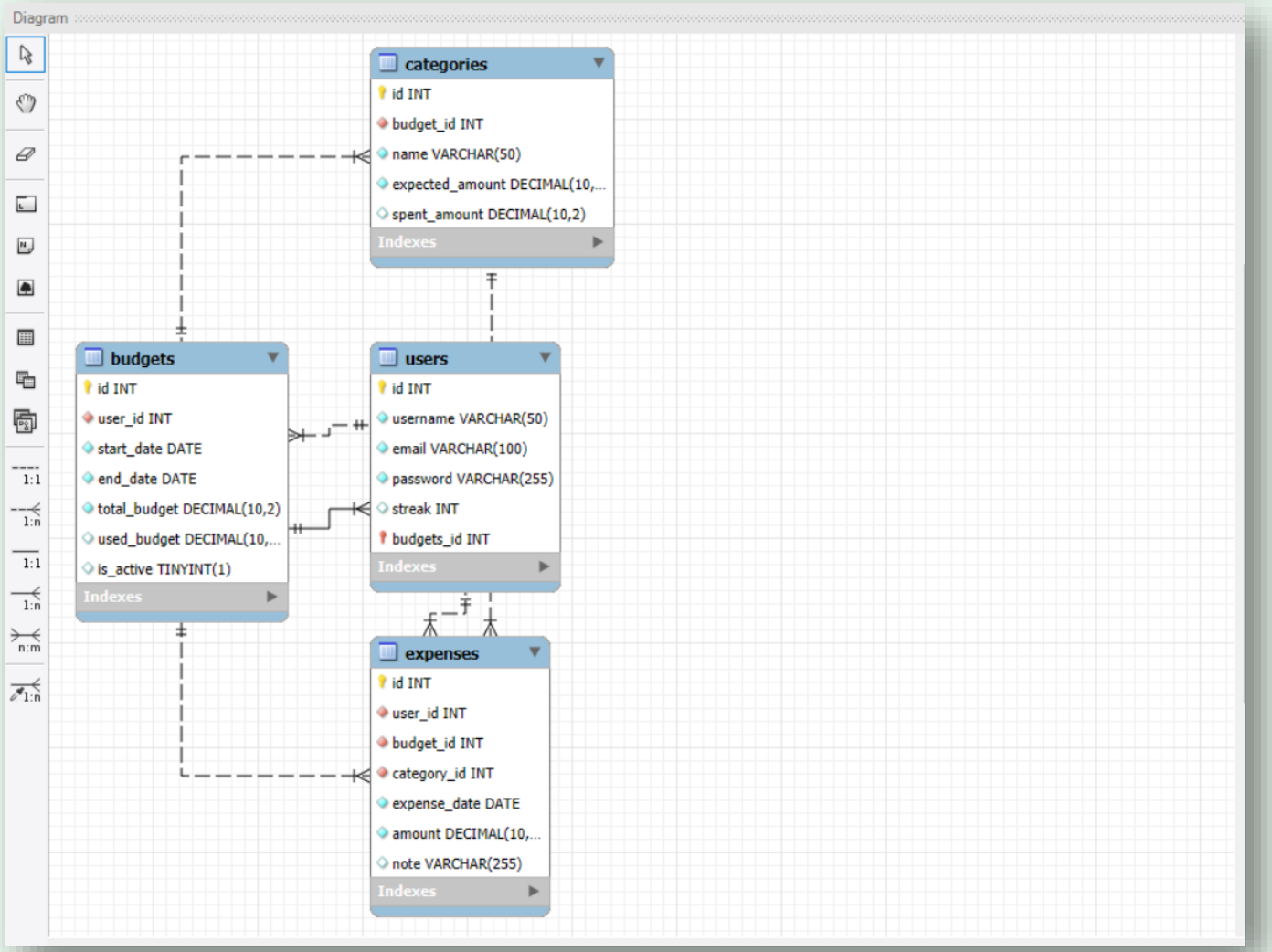
- **MySQL** running on localhost:3307, database budgetbuddy.
- Persists user accounts, budget periods, category allocations, and expense records.



ER Diagram


Users
Records
Groups
Expenses
Id
Username
Email
Password
streak
budgets
Includes
Id
user_id
budget_
category_id
expense_date
amount
note
Categories
adds
Id
user_id
start_date
end_date
total_budget
used_budget
is_active
Id
budget_id
name (
expected_amount
spent_amount
classifies

Reverse Engineering Diagram



Database Schema

1. Users

Columns

i. id (INT, PK)
ii. username (VARCHAR)
iii. email (VARCHAR)
iv. password (VARCHAR)
v. streak (INT) — count of consecutive under-budget months

Notes

i. Each row is one registered user.
ii. id is referenced by all other tables to tie data back to the user.

2. budgets

Columns

i. id (INT, PK)
ii. user_id (INT, FK → users.id)
iii. start_date, end_date (DATE) — budget period
iv. total_budget (DECIMAL) — the user's planned total
v. used_budget (DECIMAL) — running sum of actual spend
vi. is_active (TINYINT) — flags the current month's budget

Relationships

i. **1 user → N budgets** (a user can have many monthly budgets, but only one active at a time).

Database Schema

3. Categories

Columns

i. id (INT, PK)
ii. budget_id (INT, FK → budgets.id)
iii. name (VARCHAR) — e.g. “Food”, “Transport”
iv. expected_amount (DECIMAL) — portion of total_budget
v. spent_amount (DECIMAL) — tally of expenses in this category

Relationships

- **1 budget → N categories** (each budget is broken into multiple categories).

4. Expenses

Columns

i. id (INT, PK)
ii. user_id (INT, FK → users.id)
iii. budget_id (INT, FK → budgets.id)
iv. category_id (INT, FK → categories.id)
v. expense_date (DATE)
vi. amount (DECIMAL)
vii. note (VARCHAR)

Relationships

i. **1 user → N expenses**
ii. **1 budget → N expenses**
iii. **1 category → N expenses**

Every expense row belongs to exactly one user, one budget period, and one category within that budget.

Normalization

Normalization of Budget Buddy Database — Up to Third Normal Form (3NF)

Normalization is a database design technique to eliminate redundancy and improve data integrity. The Budget Buddy schema follows **3NF**, as explained below:

❖ **First Normal Form (1NF) – Atomic Columns**

Each table has:

Atomic (indivisible) values
Unique rows with a primary key

Examples:

- username, email are single-valued in users
- note in expenses is a single text field

❖ **Second Normal Form (2NF) – No Partial Dependencies**

All **non-key columns depend on the entire primary key**
Applies mainly to tables with composite keys (not used here, but still respected)

Examples:

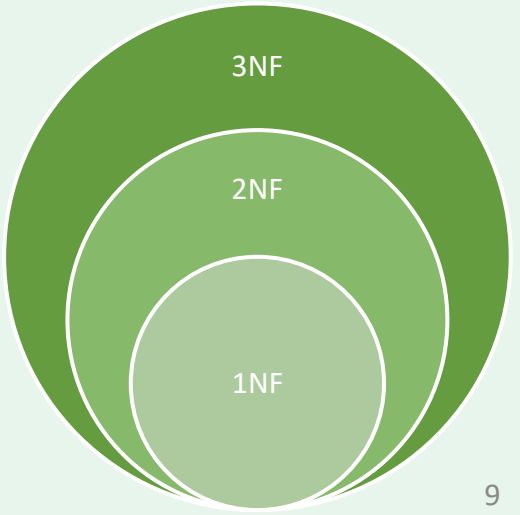
- In categories, name and expected_amount depend entirely on budget_id
- In expenses, amount, note, etc., depend on the whole row identified by id

❖ **Third Normal Form (3NF) – No Transitive Dependencies**

Non-key columns do not depend on other non-key columns

Examples:

- In users, email doesn't depend on username; both depend only on id
- In budgets, total_budget, used_budget, and is_active depend only on id, not on user_id
- In expenses, amount and note are directly related to the expense entry, not indirectly through category or user



Budget Buddy – Basic Flow

Budget Buddy

A Personal Monthly Budget Tracker

1.User Registration
→ New users sign up with name, email, and password.

2.User Login
→ Registered users log in securely.

3.Dashboard Access
→ View all personal budgets and current progress.

4.Create Budget
→ Add budgets by category, amount, and time range.

5.Add Expenses
→ Log daily expenses under selected budget.

6.Track Progress
→ See usage, remaining budget, and alerts.

7.View History
→ See list of past budgets and spending.

8.Monthly Reset
→ Budgets auto-renew or reset monthly (if opted).

9.Profile Management
→ Edit personal info and manage password.

10.View Reports
→ Visual charts for expense analysis.

Budget Buddy Flow

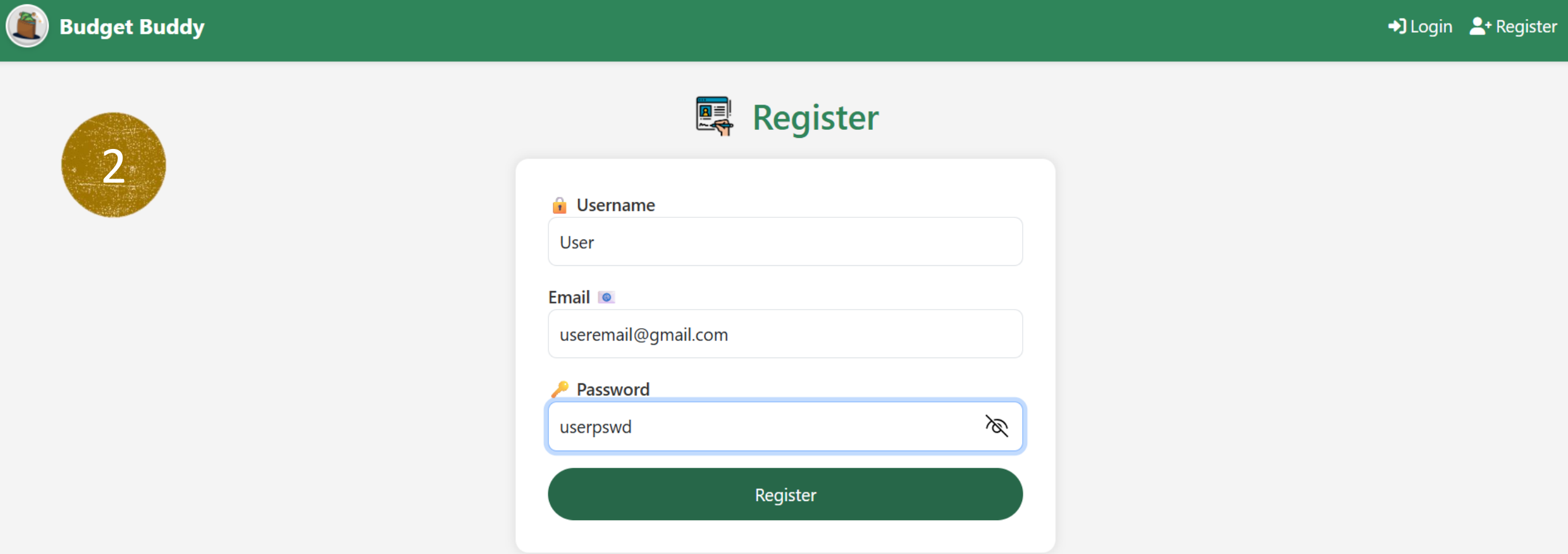
1

Budget Buddy

A Personal Monthly Budget Tracker

Register Login

A budget is telling your money where to go instead of wondering where it went.



1. Register

i. User visits /register.

ii. Fills in name, email, username, password.

iii. SQL Query 1 (Check duplication):

SELECT * FROM users WHERE username = %s OR email = %s

i. If exists → show error.

ii. If not:

SQL Query 2 (Insert new user):

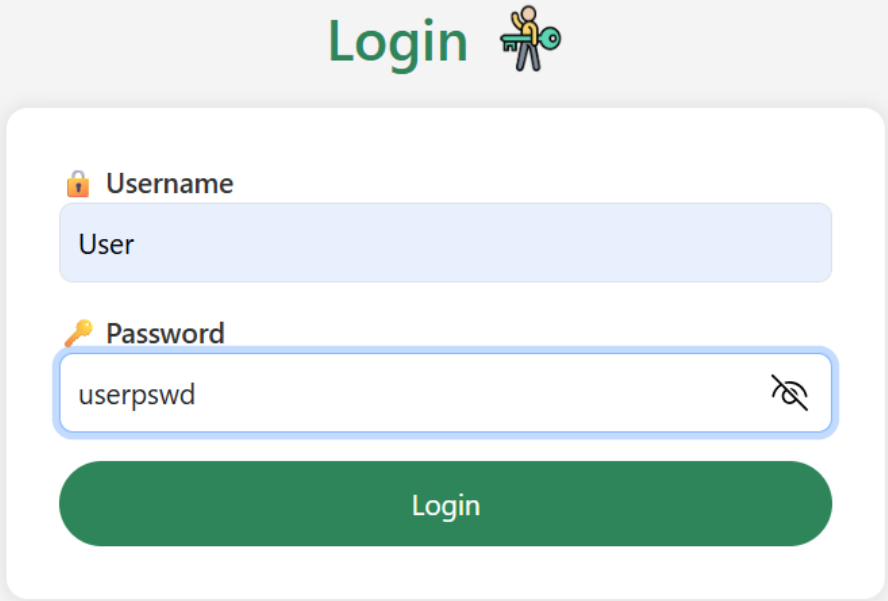
INSERT INTO users (name, email, username, password) VALUES (%s, %s, %s, %s)

i. Redirect to login page.

Register

```python
@app.route('/register', methods=['GET', 'POST'])
def register():
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        email    = request.form['email']
        password = request.form['password']
        if not username or not email or not password:
            msg = 'Please fill out all fields.'
        else:
            cursor = mysql.connection.cursor()
            cursor.execute("SELECT * FROM users WHERE username = %s OR email = %s", (username, email))
            if cursor.fetchone():
                msg = 'Account already exists!'
            else:
                hashed = generate_password_hash(password)
                cursor.execute(
                    "INSERT INTO users (username, email, password, streak) VALUES (%s, %s, %s, 0)",
                    (username, email, hashed)
                )
                mysql.connection.commit()
                msg = 'You have successfully registered!'
    return render_template('register.html', msg=msg)
```

3

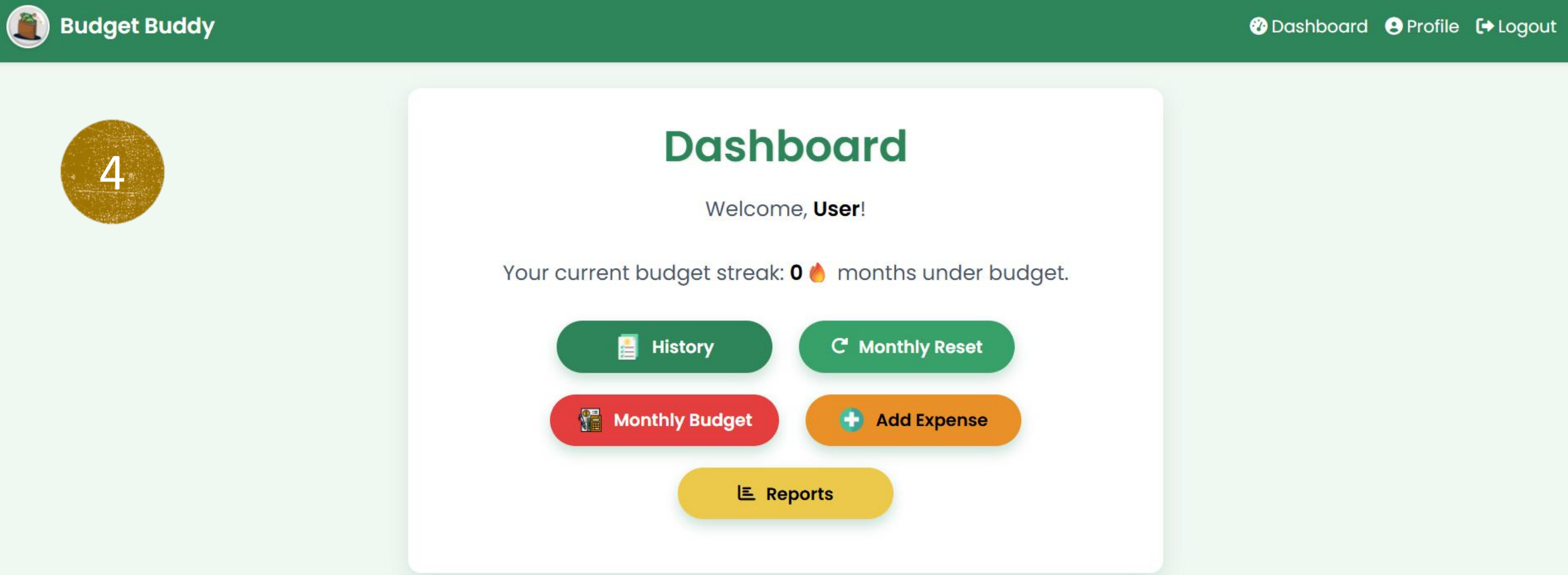


2. Login

i. User visits /login.
ii. Enters username and password.
iii. SQL Query (Fetch user):
SELECT * FROM users WHERE username = %s
i. If password matches → set session.
ii. Redirect to dashboard.

Login

```python
@app.route('/login', methods=['GET', 'POST'])
def login():
    msg = ''
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        cursor = mysql.connection.cursor()
        cursor.execute("SELECT * FROM users WHERE username = %s", (username,))
        account = cursor.fetchone()
        if account and check_password_hash(account[3], password):
            session['user_id'] = account[0]
            session['username'] = account[1]
            return redirect(url_for('dashboard'))
        else:
            msg = 'Invalid username or password.'
    return render_template('login.html', msg=msg)
```



- **Get current streak**

  SELECT streak FROM users WHERE id = %s;

  Why: Show the user how many successful cycles in a row they've kept under budget.

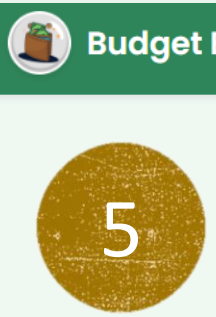
Dashboard

- **Get current streak**

  SELECT streak FROM users WHERE id = %s;

  Why: Show the user how many successful cycles in a row they've kept under budget.

```python
@app.route('/dashboard')
@login_required
def dashboard():
    user_id = session['user_id']
    cursor = mysql.connection.cursor()
    cursor.execute("SELECT streak FROM users WHERE id = %s", (user_id,))
    streak = cursor.fetchone()[0] or 0
    return render_template('dashboard.html', streak=streak)
```


Budget Buddy
Dashboard
Profile
Logout
5
Monthly Budget Setup
Start Date
01-01-2025
End Date
01-02-2025
Total Budget
50000
Categories
Category
Expected (%)
Rent
40
Groceries
20
Transport
10
Entertainment
15
Savings
15
Add Category
Save Budget

Monthly budget Setup

•**Find any previous active budget**
SELECT total_budget, used_budget FROM budgets WHERE user_id = %s AND is_active = TRUE;
Why: Determine if there was an ongoing budget to decide whether the user "succeeded" (used ≤ total) and hence increment or reset their streak
.
•**Fetch current streak**
SELECT streak FROM users WHERE id = %s;
Why: You need their existing streak count so you can bump it up (or zero it) based on last cycle's performance.

•**Update streak**
UPDATE users SET streak = %s WHERE id = %s;
Why: Store the recalculated streak back into the user record.

•**Deactivate old budgets**
UPDATE budgets SET is_active = FALSE WHERE user_id = %s;
Why: Mark any prior budgets "closed," so only one cycle is active at a time.

•**Insert new budget**
INSERT INTO budgets (user_id, start_date, end_date, total_budget, is_active, used_budget) VALUES (%s, %s, %s, %s, TRUE, 0);
Why: Create the fresh cycle with zero spent so far.

•**Insert each category**
INSERT INTO categories (budget_id, name, expected_amount, spent_amount) VALUES (%s, %s, %s, 0);
Why: Save every line-item of the user's budget plan, with the target amount and zero spent yet.

Monthly budget Setup

Check for existing “Others”

SELECT id, expected_amount FROM categories WHERE budget_id = %s AND name = 'Others';

Why: Decide whether to add leftover percentage to an existing “Others” or create a new “Others” row. r.

Update or insert “Others”

-- If exists: UPDATE categories SET expected_amount = %s WHERE id = %s; -- Otherwise: INSERT INTO categories (budget_id, name, expected_amount, spent_amount) VALUES (%s, 'Others', %s, 0);

Why: Ensure any un-allocated portion of the budget lives in “Others” so 100% of funds are accounted fo

```
# 4) Update user streak from last active budget
cursor.execute("""
    SELECT total_budget, used_budget
    FROM budgets
    WHERE user_id = %s AND is_active = TRUE
""", (user_id,))
last = cursor.fetchone()
if last:
    old_total, old_used = last
    cursor.execute("SELECT streak FROM users WHERE id = %s", (user_id,))
    current_streak = cursor.fetchone()[0] or 0
    new_streak = current_streak + 1 if old_used <= old_total else 0
    cursor.execute("UPDATE users SET streak = %s WHERE id = %s", (new_streak, user_id))
    mysql.connection.commit()

# 5) Deactivate previous budgets
cursor.execute("UPDATE budgets SET is_active = FALSE WHERE user_id = %s", (user_id,))
mysql.connection.commit()

# 6) Create new budget
cursor.execute(
    "INSERT INTO budgets (user_id, start_date, end_date, total_budget, is_active, used_budget) "
    "VALUES (%s, %s, %s, %s, TRUE, 0)",
    (user_id, start_date, end_date, total_budget)
)
```


6

Budget Buddy
Dashboard
Profile
Logout
Add Expense
Start Date
01-01-2025
End Date
01-02-2025
Load Categories
Category
Amount
Note
Rent
20000
Note (optional)
Groceries
10000
Note (optional)
Transport
4000
Note (optional)
Entertainment
9000
Note (optional)
Savings
6500
Note (optional)
Add Expense

Add Expense

1.Locate the relevant active budget
SELECT id FROM budgets WHERE user_id = %s AND is_active = TRUE AND start_date <= %s AND end_date >= %s;
Why: Confirm there's an open cycle covering the date the user is logging expenses.

2.Fetch its categories
SELECT id, name FROM categories WHERE budget_id = %s;
Why: Know which category IDs to use when inserting each expense line.

3.Insert each expense
INSERT INTO expenses (user_id, budget_id, category_id, expense_date, amount, note) VALUES (%s, %s, %s, %s, %s, %s);
Why: Record the actual spend amount, date, and any note for audit and reporting.

4.Add amount to category's spent total
UPDATE categories SET spent_amount = spent_amount + %s WHERE id = %s;
Why: Keep a running tally for each category so you can quickly show "spent vs. expected."

5.Add amount to budget's used total
UPDATE budgets SET used_budget = used_budget + %s WHERE id = %s;
Why: Update the overall monthly spend so the dashboard and "streak" logic stay accurate.

Add Expense

```python
def get_active_budget_and_categories(user_id, start_date, end_date):
    """Returns (budget_id, [(cat_id, cat_name), …]) or (None, None)."""
    cursor = mysql.connection.cursor()
    cursor.execute("""
        SELECT id FROM budgets
         WHERE user_id=%s
           AND is_active=TRUE
           AND start_date<=%s
           AND end_date>=%s
    """, (user_id, end_date, start_date))
    bud = cursor.fetchone()
    if not bud:
        return None, None

    budget_id = bud[0]
    cursor.execute(
        "SELECT id, name FROM categories WHERE budget_id=%s",
        (budget_id,)
    )
    cats = cursor.fetchall()  # list of (id, name)
    return budget_id, cats
```

```python
cursor.execute(
    "INSERT INTO expenses (user_id, budget_id, category_id, expense_date, amount, note) "
    "VALUES (%s, %s, %s, %s, %s, %s)",
    (uid, budget_id, cat_id, d_start, amount, note)
)
cursor.execute(
    "UPDATE categories SET spent_amount = spent_amount + %s WHERE id = %s",
    (amount, cat_id)
)
cursor.execute(
    "UPDATE budgets SET used_budget = used_budget + %s WHERE id = %s",
    (amount, budget_id)
)
```

Budget Progress

Total Budget: $50000.00 | Spent: $49500.00

Rent: $20000.00 / $20000.00

100.0%

Groceries: $10000.00 / $10000.00

100.0%

Transport: $4000.00 / $5000.00

80.0%

Entertainment: $9000.00 / $7500.00

120.0%

Savings: $6500.00 / $7500.00

86.7%

```python
@app.route('/budget-progress')
@login_required
def budget_progress():
    user_id = session['user_id']
    cursor = mysql.connection.cursor()
    cursor.execute("SELECT id, total_budget, used_budget FROM budgets WHERE user_id = %s AND is_active = TRUE", (user_id,))
    row = cursor.fetchone()
    if not row:
        return redirect(url_for('budget_setup'))
    budget_id, total, used = row
    cursor.execute("SELECT name, expected_amount, spent_amount FROM categories WHERE budget_id = %s", (budget_id,))
    categories = cursor.fetchall()
    return render_template('budget-progress.html', total=total, used=used, categories=categories)
```

8

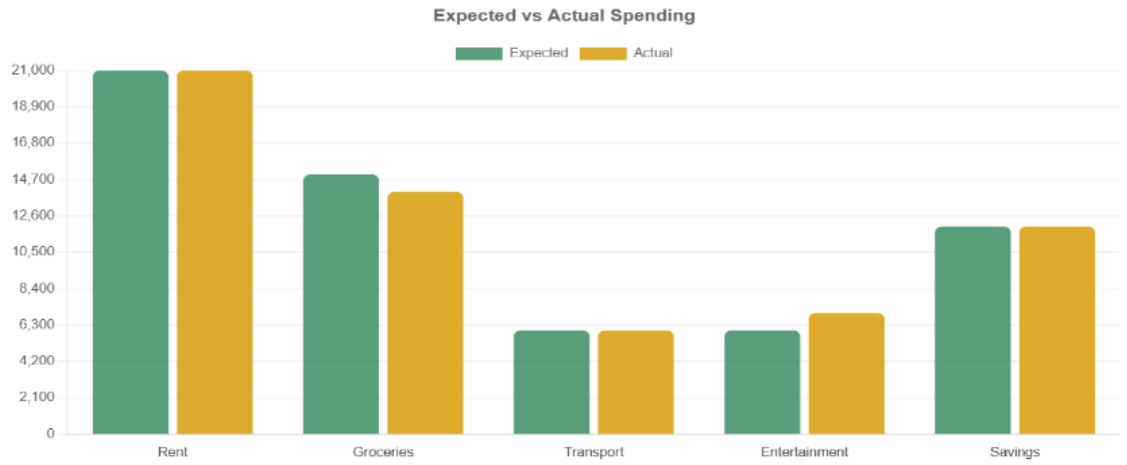
Budget History

Start Date	End Date	Total Budget	Spent	Result	Status
2025-02-01	2025-03-01	$60000.00	$60000.00	Under Budget	Active
2025-01-01	2025-02-01	$50000.00	$49500.00	Under Budget	Archived

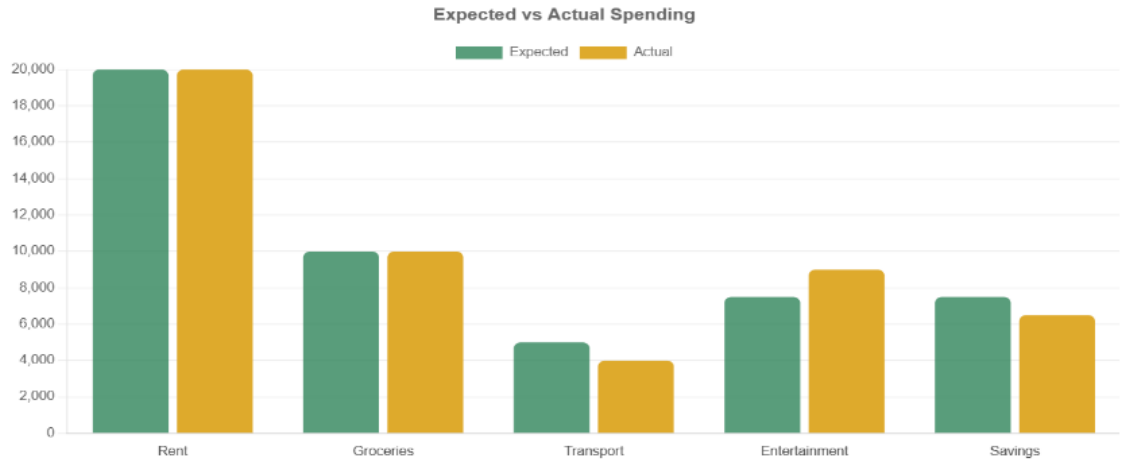
```
# Fetch all budgets (active and archived) with their status
cursor.execute(
    "SELECT start_date, end_date, total_budget, used_budget, is_active "
    "FROM budgets WHERE user_id = %s ORDER BY start_date DESC",
    (user_id,)
)
```

9

Budget Period: 2025-02-01 to 2025-03-01



Budget Period: 2025-01-01 to 2025-02-01



Reports

```
# Fetch budgets with their categories in one JOIN
cursor.execute("""
    SELECT
        b.id AS budget_id,
        b.start_date,
        b.end_date,
        c.name,
        c.expected_amount,
        c.spent_amount
    FROM budgets b
    LEFT JOIN categories c
      ON c.budget_id = b.id
    WHERE b.user_id = %s
    ORDER BY b.start_date DESC, c.id
""", (user_id,))
rows = cursor.fetchall()
```

Budget Buddy

Dashboard Profile Logout

10

Monthly Reset

Total Budget: $50000.00 | Spent: $1047.32

✅ Great job! You've stayed under budget. Your streak will increase.

Archive & Reset Current Month

Reset All Budgets

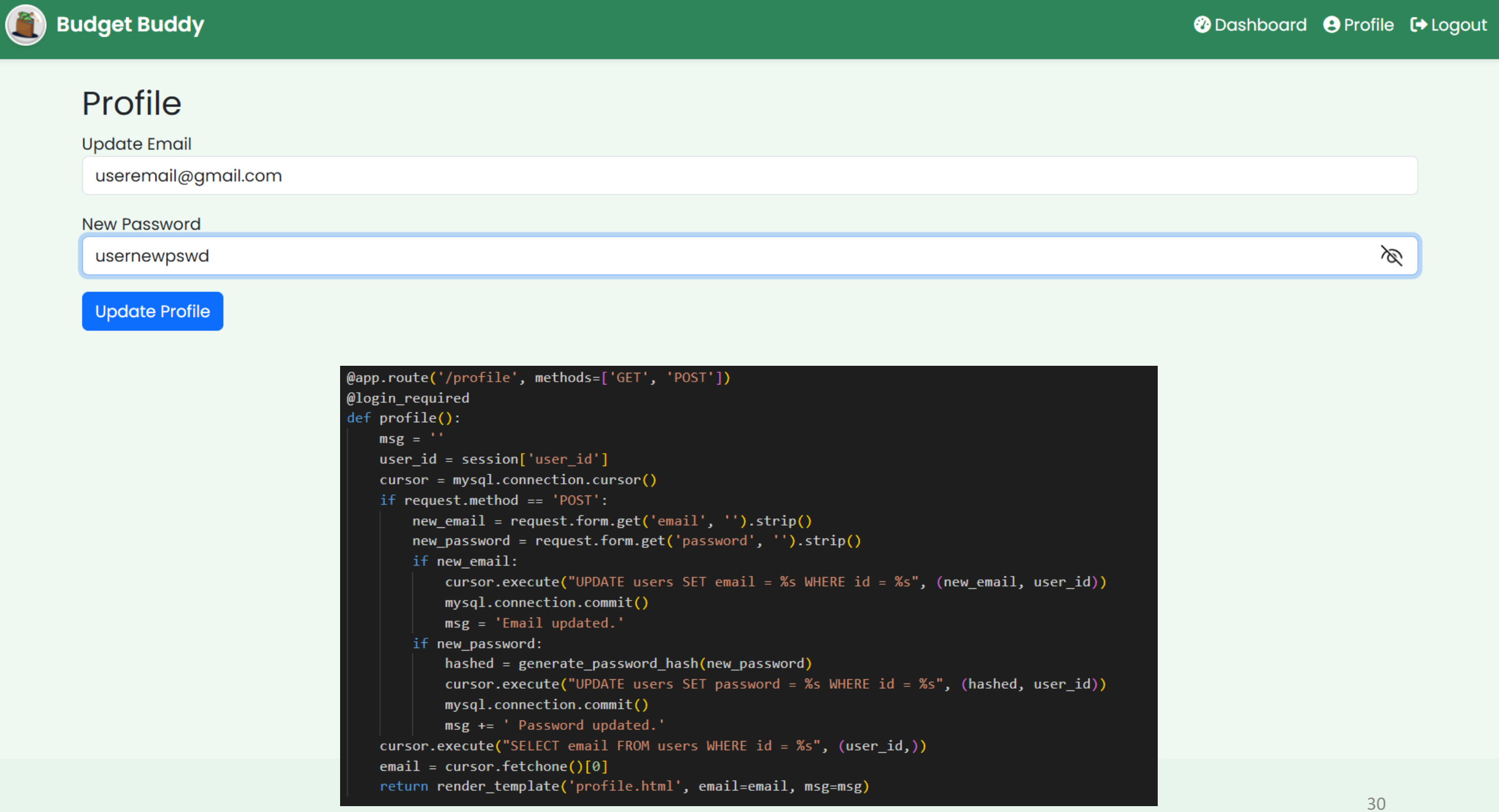
Reset

```python
@app.route('/monthly-reset', methods=['GET', 'POST'])
@login_required
def monthly_reset():
    user_id = session['user_id']
    cursor = mysql.connection.cursor()
    cursor.execute("SELECT id, total_budget, used_budget FROM budgets WHERE user_id = %s AND is_active = TRUE", (user_id,))
    budget = cursor.fetchone()
    if not budget:
        flash("No active budget found to reset.")
        return render_template('monthly-reset.html', total=0, used=0, no_budget=True)

    budget_id, total, used = budget
    if request.method == 'POST':
        cursor.execute("SELECT streak FROM users WHERE id = %s", (user_id,))
        current = cursor.fetchone()[0] or 0
        new_streak = current + 1 if used <= total else 0
        cursor.execute("UPDATE users SET streak = %s WHERE id = %s", (new_streak, user_id))
        cursor.execute("UPDATE budgets SET is_active = FALSE WHERE id = %s", (budget_id,))
        mysql.connection.commit()
        flash('Budget cycle archived. Streak updated.')
        return redirect(url_for('budget_setup'))
    return render_template('monthly-reset.html', total=total, used=used)
```

```python
@app.route('/reset-all', methods=['POST'])
@login_required
def reset_all():
    user_id = session['user_id']
    cursor = mysql.connection.cursor()

    # 1) Delete all expenses for this user
    cursor.execute("DELETE FROM expenses WHERE user_id = %s", (user_id,))
    # 2) Delete all categories belonging to this user's budgets
    cursor.execute("""
        DELETE c FROM categories c
        JOIN budgets b ON c.budget_id = b.id
        WHERE b.user_id = %s
    """, (user_id,))
    # 3) Delete all budgets for this user
    cursor.execute("DELETE FROM budgets WHERE user_id = %s", (user_id,))
    # 4) Reset user streak to 0
    cursor.execute("UPDATE users SET streak = 0 WHERE id = %s", (user_id,))
    mysql.connection.commit()
    flash('All budgets, categories, and expenses have been fully reset.')
    return redirect(url_for('budget_setup'))
```




```python
@app.route('/profile', methods=['GET', 'POST'])
@login_required
def profile():
    msg = ''
    user_id = session['user_id']
    cursor = mysql.connection.cursor()
    if request.method == 'POST':
        new_email = request.form.get('email', '').strip()
        new_password = request.form.get('password', '').strip()
        if new_email:
            cursor.execute("UPDATE users SET email = %s WHERE id = %s", (new_email, user_id))
            mysql.connection.commit()
            msg = 'Email updated.'
        if new_password:
            hashed = generate_password_hash(new_password)
            cursor.execute("UPDATE users SET password = %s WHERE id = %s", (hashed, user_id))
            mysql.connection.commit()
            msg += ' Password updated.'
    cursor.execute("SELECT email FROM users WHERE id = %s", (user_id,))
    email = cursor.fetchone()[0]
    return render_template('profile.html', email=email, msg=msg)
```

Challenges Faced

❑ **Database Design**
Hard to avoid data repetition. → Solved using normalization (1NF, 2NF, 3NF).

❑ **Login System**
Needed secure user accounts. → Used password hashing and unique emails.

❑ **Updating Budgets**
Updating totals after each expense was tricky. → Handled in Flask backend.


THE END