Centsible

Splitting money has never been simpler

Ahnaf Ahmed, Halil Akca, Tony Lin, Zara Amer, Justin Zeng

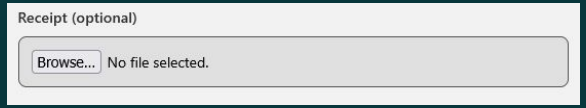
DEMO

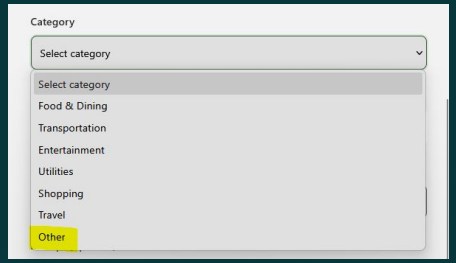
PRODUCT DEFINITION

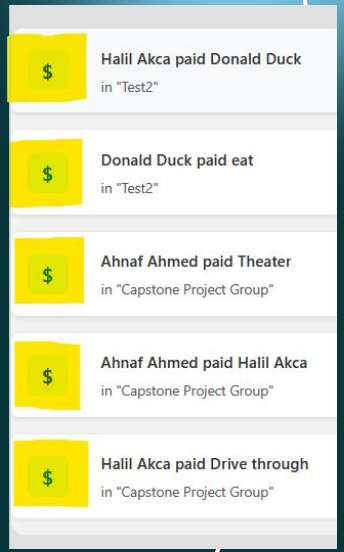
VISION:	To make splitting group expenses effortless and fair,
TARGET AUDIENCE & PROBLEM:	our product will solve the problem young adults and friend groups face when sharing costs
STRATEGY:	by giving them a simple web app with automatic balance tracking and expense logging.
GOALS:	We know our product works when we see groups easily settle debts without confusion or disputes.

Remaining Work & Stretch Goals

- Receipt scanning
- Different logos based on category
- Replace “other” with “custom” and add an additional field to describe what the category is
- Overhaul look of website and buttons
- Dark mode (just for fun, not needed)
- User Alert System
- Ability to split expenses by BOTH amount and percentage
- Export log of activities to PDF







Tech Used & Work Distribution

FRONTEND:

- Static HTML/CSS
- JavaScript

BACKEND:

- Python with Flask
- SQLite
- bcrypt
- PyJWT
- python-dotenv

How Password Hashing Was Implemented

Registration:

- User submits a password
  - hash_password()
- Hash is stored in database

Login:

- User submits email and password
- System retrieves stored hash from database
  - verify_password()
  - Returns true if they match, false otherwise

```python
def hash_password(password):
    """Hash a password using bcrypt"""
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

def verify_password(password, hashed):
    """Verify a password against its hash"""
    return bcrypt.checkpw(password.encode('utf-8'), hashed.encode('utf-8'))
```

```
balances  expenses  groups  members  payments  users
sqlite> Select * from users;
1|Halil Akca|halil@gmail.com|$2b$12$Nr373Rh7tXVC6X9KDtousOq6mFfmdpMUs6hNZAYAhOSUs8ZsIdWBq|2025-11-02T21:06:26.771346|2025-11-02T21:06:26.771346|
2|Zara Amer|zara@gmail.com|$2b$12$YTbDd.H/lDRUVZrkDcsukeL3E9..r.MwR9ZvEhDO6fxzjV4eCObGe|2025-11-02T21:10:12.187926|2025-11-02T21:10:12.187926|
3|Ahnaf Ahmed|ahnaf@gmail.com|$2b$12$f4U4Og7Rz5r83FIJVWBh28eyecwZNTKELzgiYrenVUH/vBpQvgM3EO|2025-11-02T21:11:26.934362|2025-11-02T21:11:26.934362|
4|Tony Lin|tony@gmail.com|$2b$12$ygfu2/IHTfPLfwZXGx1TaOl4VmF/Pu0HxPhGl/W/9u3rkfXGOwaWK|2025-11-02T21:12:46.012602|2025-11-02T21:12:46.012602|
5|Justin Zeng|justin@gmail.com|$2b$12$qMEU3H7rtqBalCFdl4E5iOSA2X8axu8gyoztdU3tArHODFrcGRXiq|2025-11-02T21:16:04.602429|2025-11-02T21:16:04.602429|
```

```python
@app.route('/auth/signup', methods=['POST'])
def signup():
    """
    User registration endpoint
    """
    try:
        data = request.get_json()

        # Validate required fields
        if not data or not all(k in data for k in ('name', 'email', 'password')):
            return jsonify({'error': 'Missing required fields'}), 400

        name = data['name'].strip()
        email = data['email'].strip().lower()
        password = data['password']

        # Basic validation
        if len(password) < 6:
            return jsonify({'error': 'Password must be at least 6 characters'}), 400

        if '@' not in email:
            return jsonify({'error': 'Invalid email format'}), 400
```

```python
        # Check if user already exists
        conn = get_db_connection()
        existing_user = conn.execute(
            'SELECT id FROM users WHERE email = ?', (email,)
        ).fetchone()

        if existing_user:
            conn.close()
            return jsonify({'error': 'User with this email already exists'}), 409

        # Create new user
        password_hash = hash_password(password)
        now = datetime.now().isoformat()

        cursor = conn.execute(
            'INSERT INTO users (username, email, password_hash, created_at, updated_at) VALUES (?, ?, ?, ?, ?)',
            (name, email, password_hash, now, now)
        )

        user_id = cursor.lastrowid
        conn.commit()
        conn.close()
```

```python
        # Generate token
        token = generate_token(user_id)

        return jsonify({
            'message': 'User created successfully',
            'user': {
                'id': user_id,
                'name': name,
                'email': email
            },
            'token': token
        }), 201

    except Exception as e:
        return jsonify({'error': 'Internal server error'}), 500
```

How Join Codes Work

Group Creation:

- User creates a group
  - generate_join_code()
    - Collision detection
- Code is returned to be shared

Joining a group:

- User submits 4 character join code
- Database lookup tries finding group by join_code
  - If found, user is added to members table
- Else, returns “invalid join code” error

```python
def generate_join_code():
    """Generate a unique 4-digit alphanumeric join code"""
    while True:
        # Generate 4-character code with uppercase letters and numbers
        code = ''.join(random.choices(string.ascii_uppercase + string.digits, k=4))

        # Check if code already exists
        conn = get_db_connection()
        existing = conn.execute(
            'SELECT 1 FROM groups WHERE join_code = ?', (code,)
        ).fetchone()
        conn.close()

        if not existing:
            return code
```

```python
@app.route('/api/groups', methods=['POST'])
def create_group():
    """
    Create a new group
    """
    try:
        # Get token from Authorization header
        auth_header = request.headers.get('Authorization')
        if not auth_header or not auth_header.startswith('Bearer '):
            return jsonify({'error': 'Authorization token required'}), 401

        token = auth_header.split(' ')[1]

        # Verify token
        try:
            payload = jwt.decode(token, app.config['SECRET_KEY'], algorithms=['HS256'])
            user_id = payload['user_id']
        except jwt.ExpiredSignatureError:
            return jsonify({'error': 'Token expired'}), 401
        except jwt.InvalidTokenError:
            return jsonify({'error': 'Invalid token'}), 401

        data = request.get_json()
```

```python
# Validate required fields
if not data or not data.get('group_name'):
    return jsonify({'error': 'Group name is required'}), 400

group_name = data['group_name'].strip()
group_description = data.get('group_description', '').strip()

if not group_name:
    return jsonify({'error': 'Group name cannot be empty'}), 400

conn = get_db_connection()

# Generate unique join code
join_code = generate_join_code()

# Create group
now = datetime.now().isoformat()
cursor = conn.execute(
    'INSERT INTO groups (group_name, group_description, join_code, created_by, created_at, updated_at) VALUES (?, ?, ?,
    (group_name, group_description, join_code, user_id, now, now)
)

group_id = cursor.lastrowid
```

```python
    # Add creator as member
    conn.execute(
        'INSERT INTO members (user_id, group_id, joined_at) VALUES (?, ?, ?)',
        (user_id, group_id, now)
    )

    conn.commit()
    conn.close()

    return jsonify({
        'message': 'Group created successfully',
        'group_id': group_id,
        'group_name': group_name,
        'group_description': group_description,
        'join_code': join_code,
        'created_by': user_id
    }), 201

except Exception as e:
    return jsonify({'error': 'Internal server error'}), 500
```

```python
def join_group():
    # Validate required fields
    if not data or not data.get('join_code'):
        return jsonify({'error': 'Join code is required'}), 400

    join_code = data['join_code'].strip().upper()

    if len(join_code) != 4:
        return jsonify({'error': 'Join code must be 4 characters'}), 400

    conn = get_db_connection()

    # Find group by join code
    group = conn.execute(
        'SELECT group_id, group_name, group_description FROM groups WHERE join_code = ? AND deleted_at IS NULL',
        (join_code,)
    ).fetchone()

    if not group:
        conn.close()
        return jsonify({'error': 'Invalid join code'}), 404

    group_id = group['group_id']
```

```python
# Check if user is already a member
existing_member = conn.execute(
    'SELECT 1 FROM members WHERE user_id = ? AND group_id = ? AND deleted_at IS NULL',
    (user_id, group_id)
).fetchone()

if existing_member:
    conn.close()
    return jsonify({'error': 'You are already a member of this group'}), 409

# Add user to group
now = datetime.now().isoformat()
conn.execute(
    'INSERT INTO members (user_id, group_id, joined_at) VALUES (?, ?, ?)',
    (user_id, group_id, now)
)
```

Summary

- Improved coding, teamwork, and project management
- Faced debugging & interface challenges
- Each obstacle helped us improve our processes and the app
- Continue to make cost-sharing easier, smoother, and more transparent

From the good people at

Centsible

Thank You!

Find our GitHub at this link:
https://github.com/Halila2727/ExpenseSplittingTracking