Broke Together – Backend Design Document

*Architecture: Clean Architecture using ASP.NET Core Web API & Class Libraries*

# 1. Project Overview

**BrokeTogether** is a household expense-sharing app that allows users to:

* Create or join a household.
* Track contributions made by each member.
* Maintain a shared shopping list.
* View balances and settle debts (“Balance the Pot”).

# 2. Clean Architecture Layers

We’ll use **class libraries** for separation of concerns.

BrokeTogether/

├── BrokeTogether.Api # Presentation Layer (Controllers, Filters, DI setup)

├── BrokeTogether.Application # Business Logic Layer (Services, DTOs, Validators)

├── BrokeTogether.Domain # Core Domain (Entities, Enums, Interfaces)

├── BrokeTogether.Infrastructure # Data Access (EF Core, Repositories, Migrations)

└── BrokeTogether.Tests # Unit & Integration Tests

# 3. Domain Model

### Entities

1. **User**
   * Id (Guid)
   * FullName
   * Email
   * PasswordHash
   * CreatedAt
2. **Home**
   * Id (Guid)
   * Name
   * InviteCode
   * CreatedAt
3. **HomeMember**
   * Id (Guid)
   * UserId (FK → User)
   * HomeId (FK → Home)
   * Role (Admin/Member)
4. **Contribution**
   * Id (Guid)
   * HomeId (FK → Home)
   * PaidBy (FK → User)
   * Amount (decimal)
   * Description
   * CreatedAt
5. **ShoppingListItem**
   * Id (Guid)
   * HomeId (FK → Home)
   * Name
   * Status (ToBuy / Bought)
   * BoughtBy (FK → User)
   * Amount (decimal, optional)
   * CreatedAt

## 4. Relationships

* **User ↔ Home**: Many-to-many (via **HomeMember**).
* **Home ↔ Contribution**: One-to-many.
* **Home ↔ ShoppingListItem**: One-to-many.

## 5. ERD

A diagram of a user

AI-generated content may be incorrect.

# 6. API Endpoints

### Auth

* **POST** /api/auth/register
* **POST** /api/auth/login

### Home

* **POST** /api/homes → Create Home
* **POST** /api/homes/join → Join Home with Invite Code
* **GET** /api/homes/{id} → Get Home Details

### Members

* **GET** /api/homes/{id}/members
* **POST** /api/homes/{id}/invite → Share Invite Code

### Contributions

* **POST** /api/contributions → Add Contribution
* **GET** /api/contributions/{homeId} → Get All Contributions for Home

### Shopping List

* **POST** /api/shopping → Add Item
* **PUT** /api/shopping/{id} → Update Item (Mark as Bought, Add Amount & Buyer)
* **GET** /api/shopping/{homeId} → Get Items

# 7. Clean Architecture Flow

1. **Controllers** (API layer) receive requests.
2. **Services** (Application layer) handle business logic.
3. **Repositories** (Infrastructure layer) communicate with the database.
4. **Domain Models** define entities and rules.

# 8. Sequence Diagram – Add Contribution

sequenceDiagram

participant User

participant API

participant Service

participant Repository

User->>API: POST /api/contributions

API->>Service: Validate & Process Contribution

Service->>Repository: Save Contribution

Repository-->>Service: Success

Service-->>API: Return Response

API-->>User: 201 Created

# 9. Security

* **Authentication**: JWT-based authentication.
* **Authorization**: Role-based (Admin/Member).
* **Password**: Hashed with ASP.NET Identity.

# 10.API

Perfect — creating a **detailed API plan** now will make the backend implementation much smoother.

Here’s a breakdown for **BrokeTogether** (Household budgeting + shopping list) based on the current UI:

## 1. Authentication

We need basic auth for signup/login.

### POST /api/auth/signup

**Purpose:** Create a new user.

**Request:**

{

"fullName": "Abdullah Tariq",

"email": "abdullah@email.com",

"password": "strongpassword123"

}

**Response:**

{

"userId": "u1",

"token": "jwt-token"

}

### POST /api/auth/login

**Purpose:** Login user.

**Request:**

{

"email": "abdullah@email.com",

"password": "strongpassword123"

}

**Response:**

{

"userId": "u1",

"token": "jwt-token"

}

## 2. Household Management

### POST /api/households

**Purpose:** Create a new household.

**Request:**

{

"name": "Elm Street House"

}

**Response:**

{

"householdId": "h1",

"name": "Elm Street House",

"inviteCode": "AB6X7D"

}

### POST /api/households/join

**Purpose:** Join an existing household using invite code.

**Request:**

{

"inviteCode": "AB6X7D"

}

**Response:**

{

"householdId": "h1",

"name": "Elm Street House"

}

### GET /api/households/{householdId}

**Purpose:** Get household details + members.

**Response:**

{

"householdId": "h1",

"name": "Elm Street House",

"members": [

{ "userId": "u1", "name": "Abdullah", "balance": 50 },

{ "userId": "u2", "name": "Mustafa", "balance": -25 }

],

"inviteCode": "AB6X7D"

}

## 3. Contribution / Expense Management

### POST /api/contributions

**Purpose:** Add a contribution (expense).

**Request:**

{

"householdId": "h1",

"description": "Weekly Groceries",

"amount": 75.00,

"paidBy": "u1",

"splitBetween": ["u1", "u2", "u3"]

}

**Response:**

{

"contributionId": "c1",

"description": "Weekly Groceries",

"amount": 75.00,

"paidBy": "u1",

"splitBetween": ["u1", "u2", "u3"],

"createdAt": "2025-08-04T12:00:00Z"

}

### GET /api/contributions/{householdId}

**Purpose:** Get all contributions for a household.

**Response:**

[

{

"contributionId": "c1",

"description": "Weekly Groceries",

"amount": 75.00,

"paidBy": "u1",

"splitBetween": ["u1", "u2", "u3"],

"createdAt": "2025-08-04T12:00:00Z"

}

]

## 4. Shopping List

### POST /api/shopping

**Purpose:** Add a shopping item.

**Request:**

{

"householdId": "h1",

"name": "Milk"

}

**Response:**

{

"itemId": "s1",

"name": "Milk",

"status": "to-buy"

}

### PATCH /api/shopping/{itemId}

**Purpose:** Update shopping item (mark as bought, edit name).

**Request:**

{

"status": "bought",

"amount": 20.00,

"paidBy": "u2"

}

**Response:**

{

"itemId": "s1",

"name": "Milk",

"status": "bought",

"amount": 20.00,

"paidBy": "u2"

}

### GET /api/shopping/{householdId}

**Purpose:** Get all shopping items for a household.

**Response:**

{

"toBuy": [

{ "itemId": "s1", "name": "Milk" }

],

"recentlyBought": [

{ "itemId": "s2", "name": "Bread", "amount": 5.00, "paidBy": "u1" }

]

}

## 5. Balancing the Pot

### GET /api/households/{householdId}/settlement

**Purpose:** Get simplified transactions for balancing.

**Response:**

[

{ "from": "u2", "to": "u1", "amount": 25.00 },

{ "from": "u3", "to": "u1", "amount": 10.00 }

]

### PATCH /api/households/{householdId}/settlement

**Purpose:** Mark a settlement transaction as completed.

**Request:**

{

"from": "u2",

"to": "u1",

"amount": 25.00

}

**Response:**

{ "message": "Settlement completed" }

## 6. Profile

### GET /api/profile

**Purpose:** Get logged-in user’s profile.

**Response:**

{

"userId": "u1",

"name": "Abdullah",

"email": "abdullah@email.com",

"avatarUrl": "https://example.com/avatar.jpg"

}

### PATCH /api/profile

**Purpose:** Update profile.

**Request:**

{

"name": "Abdullah Tariq",

"email": "abdullah@email.com",

"avatarUrl": "https://example.com/new-avatar.jpg"

}

**Response:**

{ "message": "Profile updated successfully" }

### Tech Notes

* **Authentication:** JWT-based.
* **Middleware:** Every /api/\* except /auth requires authentication.
* **Database:**
  + **Users**
  + **Households** (with inviteCode)
  + **Contributions** (linked to household + users)
  + **ShoppingList**
* **Balancing Algorithm:** Calculate who owes who based on contributions.

### ER Diagram (Text Form)

Users

├─ userId (PK)

├─ fullName

├─ email

├─ passwordHash

├─ avatarUrl

└─ createdAt

Households

├─ householdId (PK)

├─ name

├─ inviteCode

└─ createdAt

HouseholdMembers

├─ id (PK)

├─ householdId (FK)

├─ userId (FK)

├─ balance

└─ role

Contributions

├─ contributionId (PK)

├─ householdId (FK)

├─ description

├─ amount

├─ paidBy (FK → Users)

├─ splitBetween (array)

└─ createdAt

ShoppingList

├─ itemId (PK)

├─ householdId (FK)

├─ name

├─ status

├─ amount

├─ paidBy (FK → Users)

└─ createdAt