Kids Money Manager - Architecture Documentation

Overview

Kids Money Manager is a full-stack web application for managing children's money. It allows parents to track deposits and expenses for their children, while children can view their balance and recent transactions. The application includes expense categorization and visual analytics through pie charts.

Tech Stack

Frontend

- **React 18.2.0** - UI library for building the user interface
- **Vite 5.0.8** - Build tool and development server
- **Chart.js 4.4.0** - Charting library for data visualization
- **react-chartjs-2 5.2.0** - React wrapper for Chart.js
- **CSS3** - Styling with RTL (Right-to-Left) support for Hebrew

Backend

- **Node.js** - JavaScript runtime
- **Express 4.18.2** - Web framework for RESTful API
- **MongoDB 6.3.0** - NoSQL database for data persistence
- **CORS 2.8.5** - Cross-Origin Resource Sharing middleware
- **dotenv 16.3.1** - Environment variable management

Deployment

- **Vercel** - Frontend hosting and deployment
- **Railway** - Backend hosting and deployment
- **MongoDB Atlas** - Cloud database hosting

Development Tools

- **Git** - Version control
- **GitHub** - Code repository
- **concurrently 8.2.2** - Run multiple npm scripts simultaneously

Architecture Diagram

```
graph TB
    subgraph Client["Client Browser"]
        UI[React Frontend]
        UI -->|HTTPS| Vercel
    end

    subgraph Frontend["Frontend (Vercel)"]
        Vercel[Vercel CDN]
        Vercel -->|Serves| ReactApp[React Application]
        ReactApp -->|API Calls| API
    end

    subgraph Backend["Backend (Railway)"]
```

```
        API[Express API Server]
        API -->|Connects| MongoDB
    end

    subgraph Database["Database (MongoDB Atlas)"]
        MongoDB[(MongoDB)]
        MongoDB -->|Stores| ChildrenData[Children Data]
        MongoDB -->|Stores| Transactions[Transactions]
    end

    ReactApp -->|REST API| API
    API -->|Read/Write| MongoDB

    style UI fill:#667eea
    style ReactApp fill:#764ba2
    style API fill:#10b981
    style MongoDB fill:#f59e0b
```

Data Flow

1. User Authentication Flow

```
sequenceDiagram
    participant User
    participant Frontend
    participant SessionStorage

    User->>Frontend: Click "Parent Dashboard"
    Frontend->>SessionStorage: Check login status
    SessionStorage-->>Frontend: Not logged in
    Frontend->>User: Show login form
    User->>Frontend: Enter password (2016)
    Frontend->>SessionStorage: Store login status
    Frontend->>User: Show parent dashboard
```

2. Add Transaction Flow

```
sequenceDiagram
    participant Parent
    participant Frontend
    participant API
    participant Database

    Parent->>Frontend: Fill transaction form
    Parent->>Frontend: Submit (amount, type, category)
    Frontend->>API: POST /api/transactions
    API->>Database: Fetch child data
    Database-->>API: Child data
    API->>API: Create transaction object
```

```
    API->>API: Recalculate balance
    API->>Database: Update child (balance + transactions)
    Database-->>API: Update confirmed
    API-->>Frontend: Transaction + new balance
    Frontend->>Frontend: Refresh UI
    Frontend->>Parent: Show updated balance
```

3. View Child Dashboard Flow

```
sequenceDiagram
    participant Child
    participant Frontend
    participant API
    participant Database

    Child->>Frontend: Open child view
    Frontend->>API: GET /api/children/:childId
    API->>Database: Fetch child data
    Database-->>API: Child data (balance, transactions)
    Frontend->>API: GET /api/children/:childId/expenses-by-category?days=7
    API->>Database: Fetch transactions
    Database-->>API: Transactions
    API->>API: Filter expenses by date
    API->>API: Group by category
    API-->>Frontend: Expenses by category (7 days)
    Frontend->>API: GET /api/children/:childId/expenses-by-category?days=30
    API-->>Frontend: Expenses by category (30 days)
    Frontend->>Frontend: Render charts
    Frontend->>Child: Display dashboard with charts
```

4. Data Synchronization Flow

```
sequenceDiagram
    participant Device1
    participant Device2
    participant API
    participant Database

    Device1->>API: Add transaction
    API->>Database: Save transaction
    Database-->>API: Saved
    API-->>Device1: Success

    Note over Device2: 5 seconds later (auto-refresh)
    Device2->>API: GET /api/children/:childId
    API->>Database: Fetch latest data
    Database-->>API: Updated data
    API-->>Device2: Latest balance
    Device2->>Device2: Update UI
```

System Architecture

Frontend Architecture

```
src/
├── main.jsx                # Application entry point
├── App.jsx                 # Main app component with routing
├── components/
│   ├── ParentDashboard.jsx  # Parent interface
│   ├── ParentLogin.jsx      # Login screen
│   ├── ChildView.jsx        # Child interface
│   ├── BalanceDisplay.jsx   # Balance display component
│   ├── TransactionList.jsx  # Transaction list component
│   └── ExpensePieChart.jsx  # Pie chart component
├── utils/
│   ├── api.js               # API client functions
│   └── storage.js           # (Legacy - not used)
└── styles/
    └── App.css              # Global styles
```

Backend Architecture

```
server/
├── server.js               # Express server and API routes
├── package.json            # Dependencies
└── .env                    # Environment variables (not in git)
```

API Endpoints

Method	Endpoint	Description
GET	`/api/children`	Get all children data
GET	`/api/children/:childId`	Get specific child data
GET	`/api/children/:childId/transactions`	Get child's transactions
GET	`/api/children/:childId/expenses-by-category`	Get expenses grouped by category
POST	`/api/transactions`	Add new transaction
POST	`/api/reset`	Reset all data
GET	`/api/health`	Health check

Data Models

Child Document

```
{
  _id: "child1" | "child2",
```

```
  name: "ג'ון חיים שלי" | "אדם חיים שלי",
  balance: 0,  // Calculated from transactions
  transactions: [
    {
      id: "uuid",
      date: "2024-01-15T10:30:00.000Z",
      type: "deposit" | "expense",
      amount: 50.00,
      description: "תיאור",
      category: "אחר" | "בילויים" | "בגדים" | "ממתקים" | "משחקים" | null,
      childId: "child1" | "child2"
    }
  ]
}
```

Transaction Object

```
{
  id: "uuid",
  date: "ISO 8601 timestamp",
  type: "deposit" | "expense",
  amount: number,
  description: string,
  category: string | null,  // Only for expenses
  childId: "child1" | "child2"
}
```

Key Features

1. Parent Dashboard

- View balances for both children
- Add deposits (money in)
- Record expenses with categories
- View all transactions
- Reset all data (with confirmation)

2. Child Dashboard

- View personal balance
- View recent transactions (last 15)
- View expense pie charts:
  - Last 7 days
  - Last 30 days
- Auto-refresh every 5 seconds

3. Expense Categories

- משחקים (Games)
- ממתקים (Candy)
- בגדים (Clothes)
- בילויים (Entertainment)

- אחר (Other)

Security

- **Parent Access**: Password-protected (2016)
- **Session Storage**: Login state stored in sessionStorage (cleared on browser close)
- **CORS**: Configured to allow requests from Vercel frontend
- **Input Validation**: Server-side validation for all inputs
- **MongoDB Security**: Network access restricted, authentication required

Deployment Architecture

```
graph LR
    subgraph Development
        Local[Local Development]
        Local -->|npm run dev| LocalServer[localhost:5173]
        Local -->|npm run dev:server| LocalAPI[localhost:3001]
    end

    subgraph Production
        GitHub[GitHub Repository]
        GitHub -->|Auto Deploy| Vercel[Vercel Frontend]
        GitHub -->|Auto Deploy| Railway[Railway Backend]
        Railway -->|Connects| MongoDBAtlas[MongoDB Atlas]
        Vercel -->|API Calls| Railway
    end

    style Vercel fill:#000000,color:#fff
    style Railway fill:#0E0E0E,color:#fff
    style MongoDBAtlas fill:#13AA52,color:#fff
```

Environment Variables

Frontend (Vercel)

- `VITE_API_URL` - Backend API URL (e.g., `https://your-app.up.railway.app/api`)

Backend (Railway)

- `MONGODB_URI` - MongoDB connection string
- `PORT` - Server port (default: 3001)

Build Process

Frontend Build

1. Vite bundles React application
2. Output: `dist/` directory
3. Vercel serves static files from `dist/`

Backend Build

1. Railway installs dependencies (`npm install`)

2. Starts server with `npm start`
3. Server listens on `0.0.0.0:PORT`

Data Persistence

- **Storage**: MongoDB Atlas (cloud database)
- **Fallback**: In-memory storage (if MongoDB unavailable)
- **Data Structure**: One document per child in `children` collection
- **Balance Calculation**: Calculated dynamically from transactions array

Performance Considerations

- **Auto-refresh**: Child views refresh every 5 seconds
- **Caching**: No explicit caching (real-time data)
- **Database**: MongoDB indexes on `_id` for fast lookups
- **CDN**: Vercel CDN for fast static asset delivery

Error Handling

- **Frontend**: Try-catch blocks with user-friendly error messages
- **Backend**: Error middleware with detailed logging
- **API**: HTTP status codes (400, 404, 500)
- **Fallback**: Graceful degradation if MongoDB unavailable

Future Enhancements

- User authentication system
- Multiple parent accounts
- Export transactions to CSV/PDF
- Email notifications
- Mobile app version
- Advanced analytics and reports

Version History

- **v1.03** - Added expense categories and pie charts
- **v1.02** - Added reset balances button
- **v1.01** - Initial release with basic features

Last Updated: December 2024