Technical Documentation - Kids Money Manager

Executive Summary

Kids Money Manager is a full-stack web application built with React and Node.js, designed to help parents manage their children's money. The application features real-time synchronization across devices, expense categorization, and visual analytics.

Technology Stack

Frontend Stack

Technology	Version	Purpose
React	18.2.0	UI framework
Vite	5.0.8	Build tool & dev server
Chart.js	4.4.0	Data visualization
react-chartjs-2	5.2.0	React Chart.js wrapper
CSS3	-	Styling with RTL support

Backend Stack

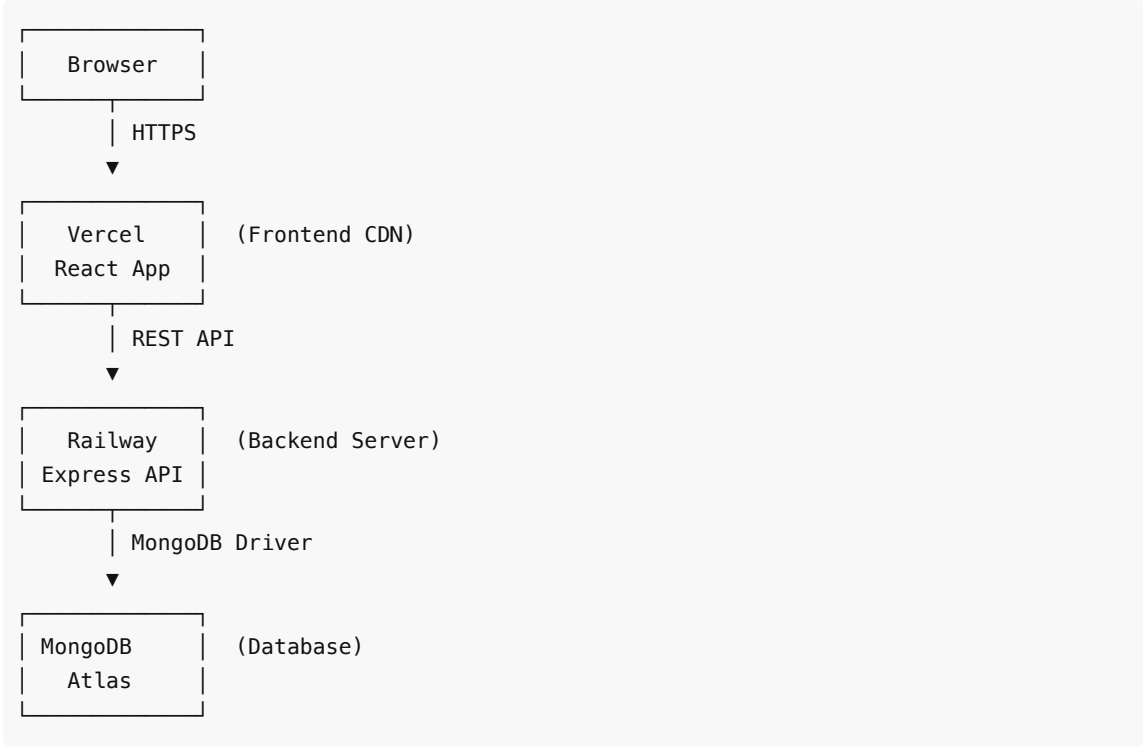
Technology	Version	Purpose
Node.js	24.12.0	Runtime environment
Express	4.18.2	Web framework
MongoDB	6.3.0	Database driver
CORS	2.8.5	Cross-origin support
dotenv	16.3.1	Environment variables

Infrastructure

Service	Purpose	Plan
Vercel	Frontend hosting	Free
Railway	Backend hosting	Paid
MongoDB Atlas	Database hosting	Free (M0)
GitHub	Source control	Free

System Architecture

High-Level Architecture



Component Structure

```
Frontend (React)
├── App.jsx (Main router)
├── Components/
│   ├── ParentDashboard.jsx
│   ├── ParentLogin.jsx
│   ├── ChildView.jsx
│   ├── BalanceDisplay.jsx
│   ├── TransactionList.jsx
│   └── ExpensePieChart.jsx
└── Utils/
    └── api.js (API client)

Backend (Express)
└── server.js
    ├── Middleware (CORS, JSON parser)
    ├── Database connection
    └── API Routes
        ├── GET /api/children
        ├── GET /api/children/:id
        ├── GET /api/children/:id/transactions
        ├── GET /api/children/:id/expenses-by-category
        ├── POST /api/transactions
        └── POST /api/reset
```

Data Models

Child Document Structure

```
{
  _id: "child1" | "child2",           // Primary key
  name: "אדם חיים שלי",                // Child name
  balance: 150.50,                     // Current balance (calculated)
  transactions: [                      // Array of transactions
    {
      id: "550e8400-e29b-41d4-a716-446655440000",
      date: "2024-12-31T10:30:00.000Z",
      type: "expense",
      amount: 25.00,
      description: "קניית משחק",
      category: "משחקים",
      childId: "child1"
    }
  ]
}
```

Transaction Types

- **deposit**: Money added (no category)
- **expense**: Money spent (requires category)

Expense Categories

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

API Specification

Base URL

- Production: `https://your-app.up.railway.app/api`
- Development: `http://localhost:3001/api`

Endpoints

GET /api/children

Get all children data.

Response:

```
{
  "children": {
    "child1": {
      "name": "אדם חיים שלי",
      "balance": 150.50,
      "transactions": [...]
```

```
    },
    "child2": {
      "name": "ג'ון חיים שלי",
      "balance": 75.00,
      "transactions": [...]
    }
  }
}
```

GET /api/children/:childId

Get specific child data.

Parameters:

- `childId`: "child1" | "child2"

Response:

```
{
  "name": "אדם חיים שלי",
  "balance": 150.50,
  "transactions": [...]
}
```

GET /api/children/:childId/transactions

Get child's transactions.

Query Parameters:

- `limit` (optional): Number of transactions to return

Response:

```
{
  "transactions": [
    {
      "id": "uuid",
      "date": "2024-12-31T10:30:00.000Z",
      "type": "expense",
      "amount": 25.00,
      "description": "קניית משחק",
      "category": "משחקים",
      "childId": "child1"
    }
  ]
}
```

GET /api/children/:childId/expenses-by-category

Get expenses grouped by category.

Query Parameters:

- `days` (required): Number of days (7 or 30)

Response:

```
{
  "expensesByCategory": [
    {
      "category": "משחקים",
      "amount": 100.00
    },
    {
      "category": "ממתקים",
      "amount": 50.00
    }
  ],
  "totalDays": 30
}
```

POST /api/transactions

Add a new transaction.

Request Body:

```
{
  "childId": "child1",
  "type": "expense",
  "amount": 25.00,
  "description": "קנית משחק",
  "category": "משחקים"
}
```

Response:

```
{
  "transaction": {
    "id": "uuid",
    "date": "2024-12-31T10:30:00.000Z",
    "type": "expense",
    "amount": 25.00,
    "description": "קנית משחק",
    "category": "משחקים",
    "childId": "child1"
  },
  "balance": 125.50,
  "updated": true
}
```

POST /api/reset

Reset all data (balances and transactions).

Response:

```
{
  "success": true,
  "message": "All balances and transactions have been reset"
}
```

GET /api/health

Health check endpoint.

Response:

```
{
  "status": "ok",
  "db": "connected" | "memory"
}
```

Data Flow

Adding a Transaction

1. User fills form in Parent Dashboard
2. Frontend validates input
3. Frontend sends POST request to `/api/transactions`
4. Backend validates request
5. Backend fetches child from database
6. Backend creates transaction object
7. Backend adds transaction to array
8. Backend recalculates balance
9. Backend updates child document
10. Backend returns success response
11. Frontend refreshes UI
12. Other devices auto-sync (5 second interval)

Viewing Analytics

1. Child opens their view
2. Frontend requests child data
3. Frontend requests expenses for 7 days
4. Frontend requests expenses for 30 days
5. Backend filters transactions by date
6. Backend groups by category
7. Backend sums amounts per category
8. Frontend renders pie charts
9. Charts auto-refresh every 5 seconds

Security

Authentication

- Parent dashboard: Password-protected (2016)

- Session-based: Stored in browser sessionStorage
- No persistent authentication tokens

Data Protection

- MongoDB network access restrictions
- CORS configured for specific origins
- Input validation on server-side
- No sensitive data in frontend

Best Practices

- Environment variables for secrets
- HTTPS for all communications
- Error messages don't expose internals

Performance

Frontend

- React component optimization
- Efficient re-renders
- Chart.js optimized rendering
- CDN delivery (Vercel)

Backend

- Single database query per request
- In-memory calculations
- Efficient MongoDB queries
- Connection pooling

Caching

- No explicit caching (real-time data required)
- Browser caching for static assets
- CDN caching for frontend bundle

Scalability

Current Limitations

- Single MongoDB collection
- No user authentication
- Fixed number of children (2)
- In-memory fallback (not scalable)

Potential Improvements

- User accounts and authentication
- Multiple children per account
- Database indexing
- Redis caching layer
- Load balancing

Monitoring

Logging

- Server-side console logging
- Error tracking in Railway logs
- Build logs in Vercel
- Client-side error logging

Metrics

- Railway: Deployment status, logs
- Vercel: Build status, function logs
- MongoDB Atlas: Database metrics

Development Workflow

1. **Local Development**

   ```
   npm run dev:all  # Runs both frontend and backend
   ```

2. **Code Changes**
   - Edit files locally
   - Test locally
   - Commit changes
   - Push to GitHub
3. **Automatic Deployment**
   - GitHub webhook triggers Vercel
   - GitHub webhook triggers Railway
   - Both platforms build and deploy automatically
4. **Verification**
   - Check deployment logs
   - Test in production
   - Monitor for errors

Troubleshooting

Common Issues

1. **"Failed to fetch"**
   - Check VITE_API_URL in Vercel
   - Verify Railway service is running
   - Check CORS configuration
2. **Data not syncing**
   - Verify MongoDB connection
   - Check Railway logs
   - Verify API endpoints
3. **Charts not showing**

    - Check date filtering logic
    - Verify expenses exist
    - Check browser console for errors

Future Enhancements

- User authentication system
- Multiple parent accounts
- Export functionality (CSV/PDF)
- Email notifications
- Mobile app
- Advanced analytics
- Budget limits per category
- Recurring transactions

Documentation Version: 1.0
Last Updated: December 2024
Maintained By: Development Team