




Smart Ledger

Team #1

By: Blake Childress, Andrei Cozma, Hunter Price, Tyler
Beichler, Emanuel Chavez, Jacob Leonard, Lillian Coar



Problem Definition

- People need a quick, easy, and free way to split the cost and keep track of expenses between people or groups of people.
- Current solutions are either not free, not user-friendly, or lack in features.
- The primary function of Smart Ledger is to split purchases into groups.
- Possible additions include
 - Receipt scanning
 - Direct integration with companies like Venmo and PayPal
 - Analytics for businesses

Background Information

- Current expense tracking solutions are either not free, not user-friendly, or lack in features.
 - No Paying from within the app (Splitwise)
 - Essential features behind paywall (Evenfy)
 - Not on all platforms, limiting who you can add (Splittr)
 - Limited to certain types of transactions (Tab)
- Venmo and CashApp
 - Have to manually calculate split
 - Keep track of purchases with another method

Requirements Specification

Customer Requirements	Description
1. Functionality	Core features such as dashboards that accurately track spendings, group support, and splitting expenses
2. Usability	Clear visual hierarchy for optimal user experience
3. Accessibility	Work seamlessly on major platforms and devices, support multiple currencies
4. Privacy	Securing user information and monetary transactions
5. Receipts	Scan or manually enter receipt information

Requirements Specification

Engineering Requirements	Description
1. Cloud Service	Cloud service that best supports frontend and backend needs while also being cost-efficient
2. Database	Store user and group information regarding spendings
3. Security	Implement exception handling and other security protocols
4. Artificial Intelligence Tools	Reports and analytics
5. Optical Character Recognition	Scan receipts

* All requirements are design variables (DV)

Technical Approach

- Hosting Service
- Frontend
- Backend
- Database
- Communication between the frontend and backend
- Splitting algorithm



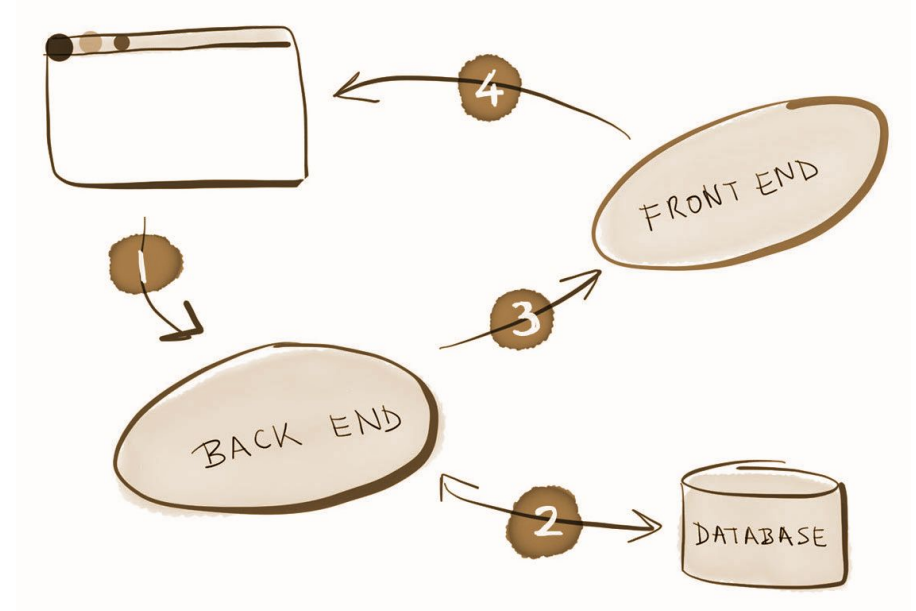
Design Decision Overview

Front End

- **Features**
 - Mobile friendly
 - Frameworks
 - Libraries

Back End

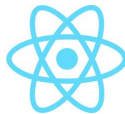
- **Use a database to store information**
 - Relational?
 - Non-relational?
- **Features:**
 - Secure
 - Reliable
 - Resilient



Design Decision Overview

- Need 4 Major Components:
 - Frontend
 - What users can see
 - Backend
 - What sends the data to the users
 - Database
 - Where is the data stored
 - Hosting Service
 - How can users get to the website

Component	Tools/Libraries
Frontend	React.js Node.js
Backend	Flask
Backend DB	MongoDB
Hosting Service	Amazon Web Services (AWS)



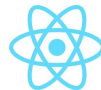
Analysis and Calcs to Support Design Selection - Front End

Criteria:

- Fast & Easy Deployment
 - Learning Curve
 - Support
- Maintainable Development
- Performance
- Familiarity

Frameworks:

- Angular
- React
- Vue



Front End Weighted Decision Matrix

Framework	Learning Curve	Familiarity	Support	Performance	Structure	Score
Angular	1	2	2	1	3	9
React	2	3	3	2	2	12
Vue	3	1	1	3	1	9

Analysis and Calcs to Support Design Selection - Back End

Cloud Services:

- Amazon Web Services (AWS)
- Google Cloud Platform (GCP)
- Microsoft Azure

Criteria:

- Familiarity
- Documentation
- Pricing
- Relevance

Databases:

- MongoDB
- CouchDB
- Couchbase Server

Criteria:

- Familiarity
- CAP Principles
 - Consistency (C)
 - Availability (A)
 - Partition tolerance (P)



Analysis and Calcs to Support Design Selection - Back End Cont.

Criteria:

- Familiarity
- Performance
- Ease Of Use
- Flexibility
- Relevance

Backend:

- PHP
- Node.js express
- Flask



Back End Weighted Decision Matrix

Cloud Service	Familiarity	Documentation	Pricing	Job Relevance	Score
AWS	3	3	2	3	11
GCP	2	2	3	2	9
Azure	1	1	1	1	4

Database	Familiarity	Consistency	Availability	Partition Tolerance	Score
MongoDB	3	3	1	3	10
CouchDB	2	1	3	1	6
Couchbase Server	1	2	2	2	8

scale 1-3

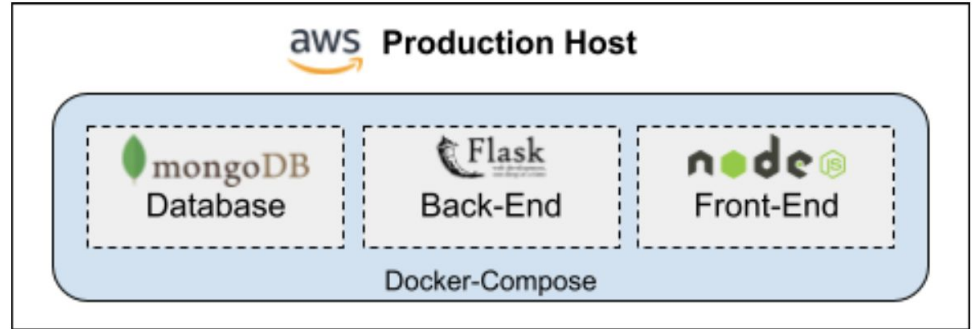
Back End Weighted Decision Matrix Cont.

Backend	Familiarity	Performance	Ease Of Use	Flexibility	Relevance	Score
PHP	1	1	1	1	1	5
Node.js w/ Express	2	3	2	2	3	12
Flask	3	2	3	3	2	13

Analysis and Calcs to Support Design/Selection

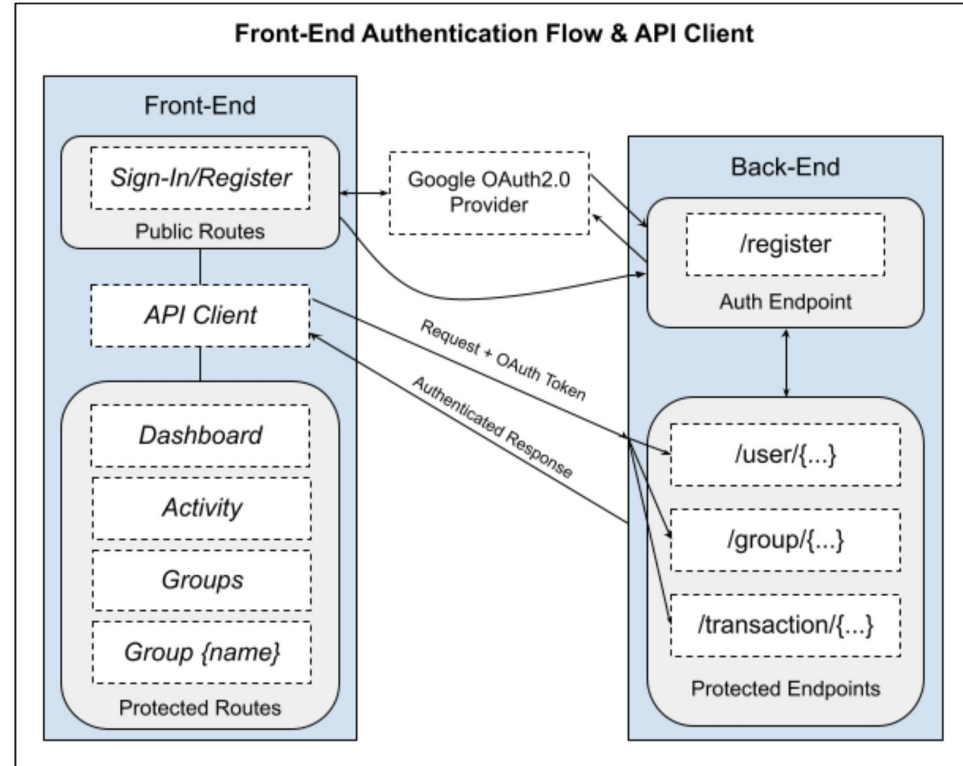
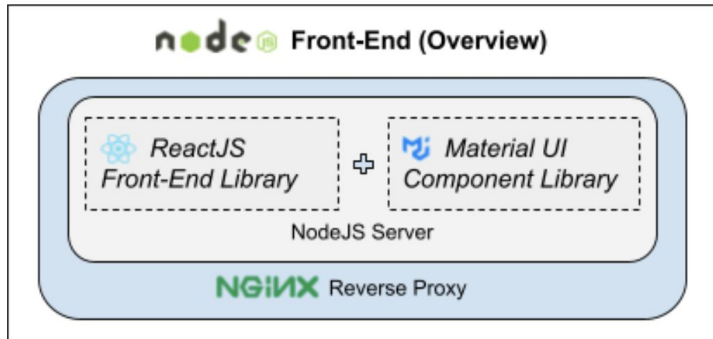
Product Architecture - Overview

- **Prototype of WebApp**
 - Production Server - AWS
 - Scalability & Security
- **Deployment Scripts**
 - Automate and describe local environment deployment
 - Production environment - w/ Docker Containers & Reverse Proxy
- **Docker Containers**
 - Front-End (Client)
 - Back-End (API)
 - Database (MongoDB)
- **Interactions Overview**
 - Client ↔ API
 - API ↔ Database

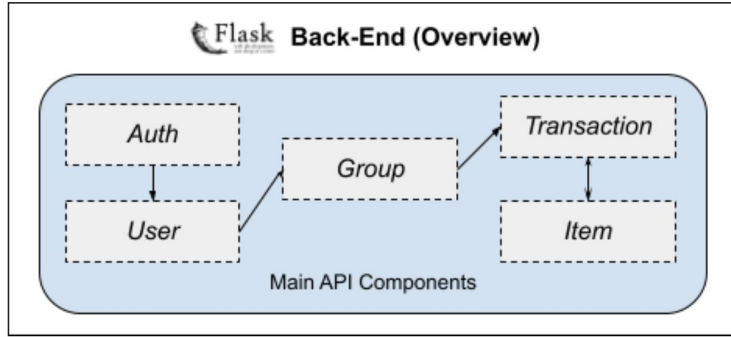


Product Architecture - Front-End

- **NodeJS Server**
 - **Front-End Lib.** - ReactJS
 - **Component Lib.** - Material UI
- **Inputs:** Responses from Back-End
- **Authentication:** Google Auth API
 - Private & Public Routes
- **Outputs:** Information Display



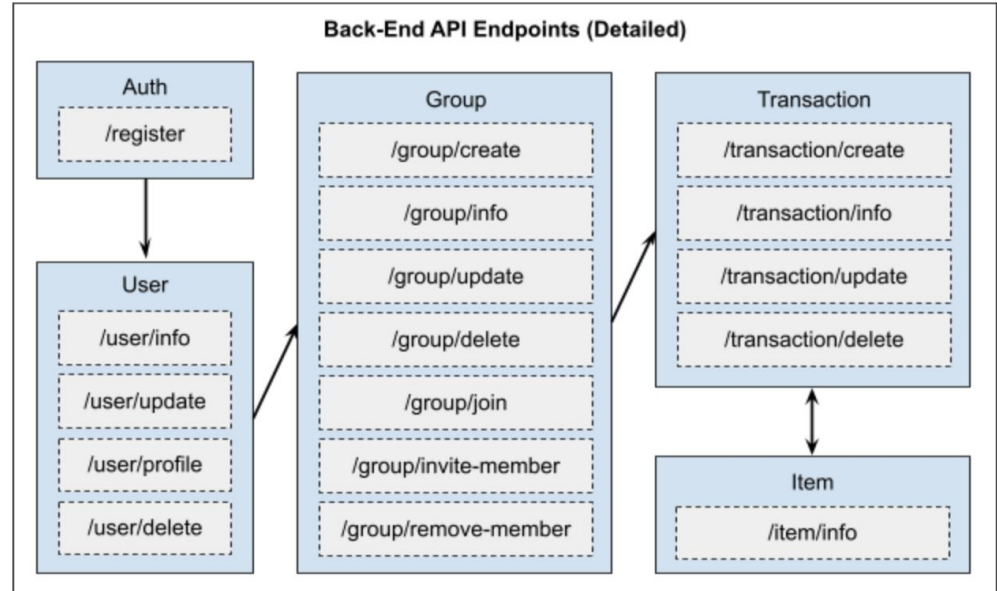
Product Architecture - Back-End



- **Inputs:** API Calls
 - Queried by the Front-End
 - Ensures user is authenticated
 - Interaction w/ Database
- **Outputs:**
 - Responses back to Front-End

API Endpoints: Separated by level of interaction

- Handling user, group, transactions, etc.



Configuration Design

- Front-end
 - Framework - React
 - Single-page Application
 - ES6
 - Components and Services
 - Google OAuth
- Back-end
 - API - Flask
 - Database - MongoDB
 - Several security measures
- Hosting
 - AWS
 - Scalable
 - Inexpensive
- Sharing Algorithm
 - Uses a shared ledger concepts
 - Uses deltas for optimized calculations
 - Allows for continuous additions and updates

Prototype Test Plan

- Built back-end, front-end, and API (React, MongoDB, PyMongo, PyTest)
 - Tied front-end to back-end with API endpoints
- Built Docker Containers for local testing
- Created API unit tests
 - Confirmed API endpoints were working properly
- Built entire application and ran tests with users and groups
- Finalized application and fixed bugs



mongoDB[®]



python[™]

Future Work

- Optical Character Recognition (OCR)
- Further development on front/back end
 - Better support for groups, integration with banks/PayPal, etc.
- Security hardening
 - Input validation, penetration testing, etc.

Budget

Item	Estimated Cost
AWS Hosting Services - Free Tier	\$20
MongoDB	\$0
Jira - Student	\$0
Discord	\$0
GitHub Student Developer Pack	\$0
Domain*	\$0
Total	\$20

DEMO