Budgeted Project

By

CS 157A - Team 12

Tuong Nguyen
Zehua Liu
Marcus Zhou

Table of Content

REQUIREMENTS

Project Description

This project will be a database application that is based on the financial domain. Our group is going to develop a robust B2C budgeting app. People nowadays oftentimes have difficulties managing their budget as well as income. Services such as Netflix, Spotify, Amazon Prime, and various mortgage payments are mostly monthly recurring, and they are becoming a huge portion of people's spending. With so many transactions taking place on a monthly basis, people fail to keep track of all of their monthly recurring spending. Without an insightful and informative understanding of their own personal finance, people tend to make poorer financial decisions. As a result, people are gradually getting into worse financial situations.

The goal of this application is to provide a comprehensive way to monitor the conditions of the budget. Users will be able to monitor their financial transactions to track their income and expenses with ease. We will also implement a user-friendly interface that is intuitive to navigate throughout the application. The constant struggle of issues with other similar applications provided the motivation to create another such application with those problems addressed

System Environment

Hardware	HP Pavilion Notebook Signature Edition Dell MacBook Air
Software	Atom, Sublime
Application Language	Javascript, SQL, HTML, CSS
Technology	ReactJS/ NodeJS/ ExpressJS MySQL / MySQL WorkBench 8.0

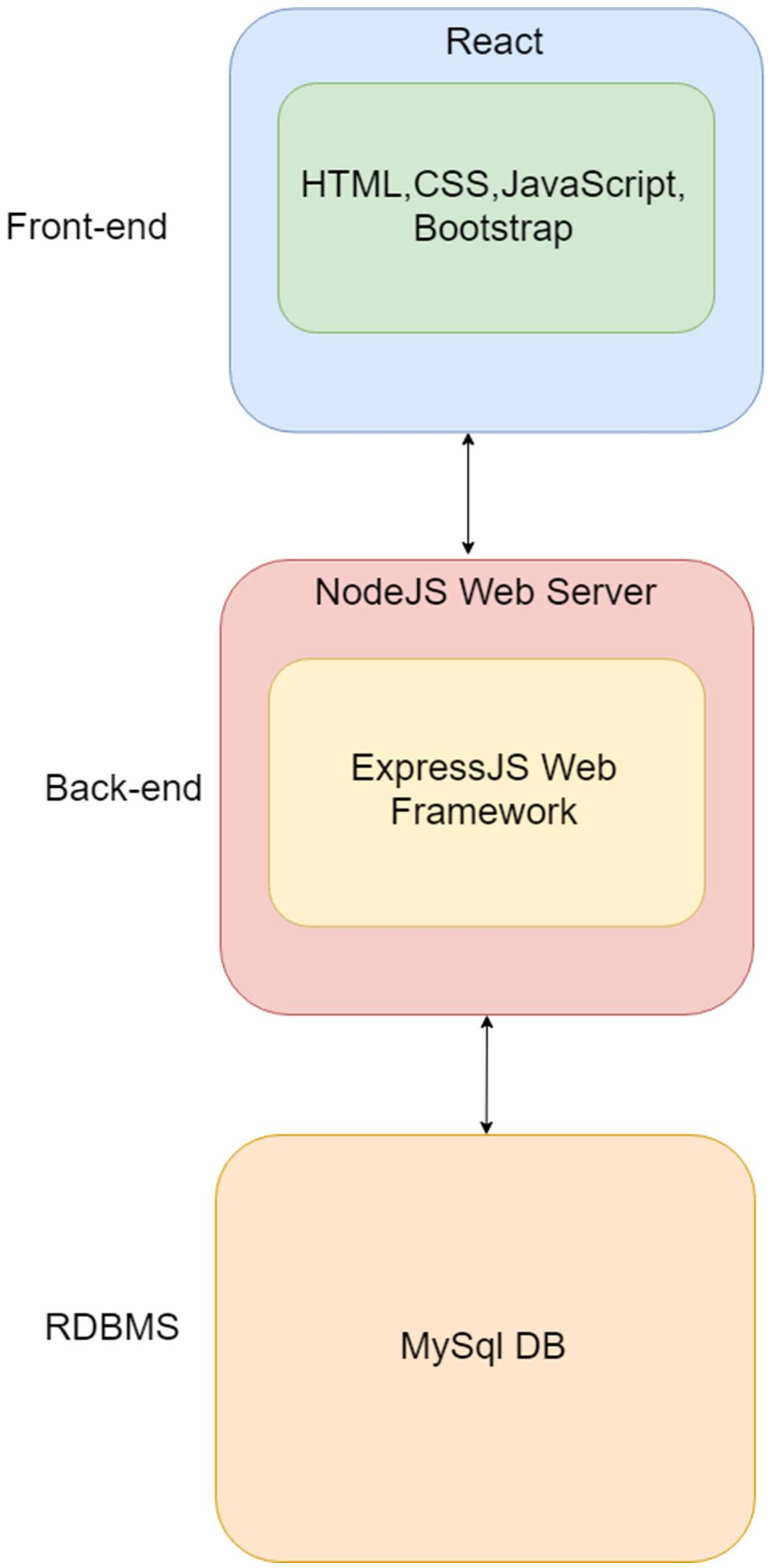


Figure 1: System Architecture

Functional Requirement
Our target audience is those who participate in economical events where they have income and expense on a regular basis. The user will be able to sign up to create an account where they will be able to make entries to record their financial transactions. They can also generate the monthly report to review the transactions they made for the month. Furthermore, the application will allow users to delete their accounts as they wish.

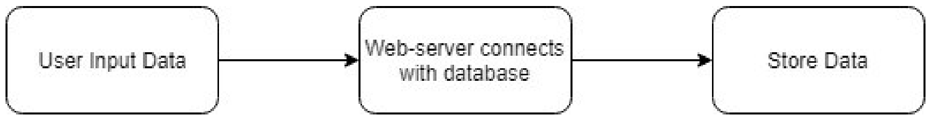


Figure 2: Functional Flow Overview

<u>User Account Creation/Login</u>
- The end-user shall be able to use the application and create an account with a password and other attributes.
- End-user shall be able to delete their account via GUI

<u>Record Income, Expense, and Saving</u>
- End-user shall be able to make entries to record their income.
- End-user shall be able to make entries to record the saving.
- Income will be classified into categories: job, investment, and bank interest, others
- Users shall describe each income transaction.
- End-user shall be able to make entries to input their expenses.
- The expense will be classified into categories: household, utility, rent, others
- End-user shall be able to describe each expense transaction.

<u>Update Transaction</u>
- The end-user shall be able to make updates and changes to the existing transactions in the database.

<u>Delete Recurring Income & Expenses</u>
- End-users shall be able to delete recurring income and expenses types.

<u>Generate Report</u>
- Users shall be able to generate a graph or chart type report based on all the data associated with their accounts.

Non-Functional Requirement
We will implement a dashboard as our graphical user interface where the user can intuitively communicate with the Budgeted application. We will also secure our application by implementing access control for users. So each user will have their own login detail and will not be able to access other accounts except their own.

<u>Dashboard</u>
- A dashboard will be implemented for the Budgeted application where users can conduct their functional activities.

Security

- Input Email will be validated to prevent spam accounts.
- After account creation password will be hashed and salted and securely stored in the database.
- Access tokens will be stored in localStorage for authentication

Access Control

- Users are authenticated using JSON web tokens (JWT)
- Each access token will expire after a week and users will have to log in again.

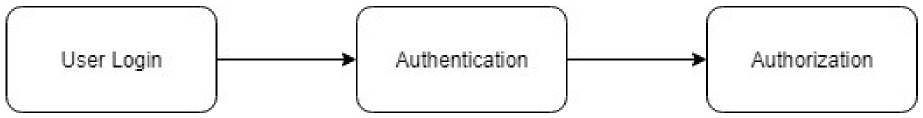


Figure 3: Access Control Overview

DESIGN

Entity-Relationship Diagram
We identified 7 entities and 4 relation tables for Budgeted. 6 entities are users, budget, income, expenses, transactions, and analysis.

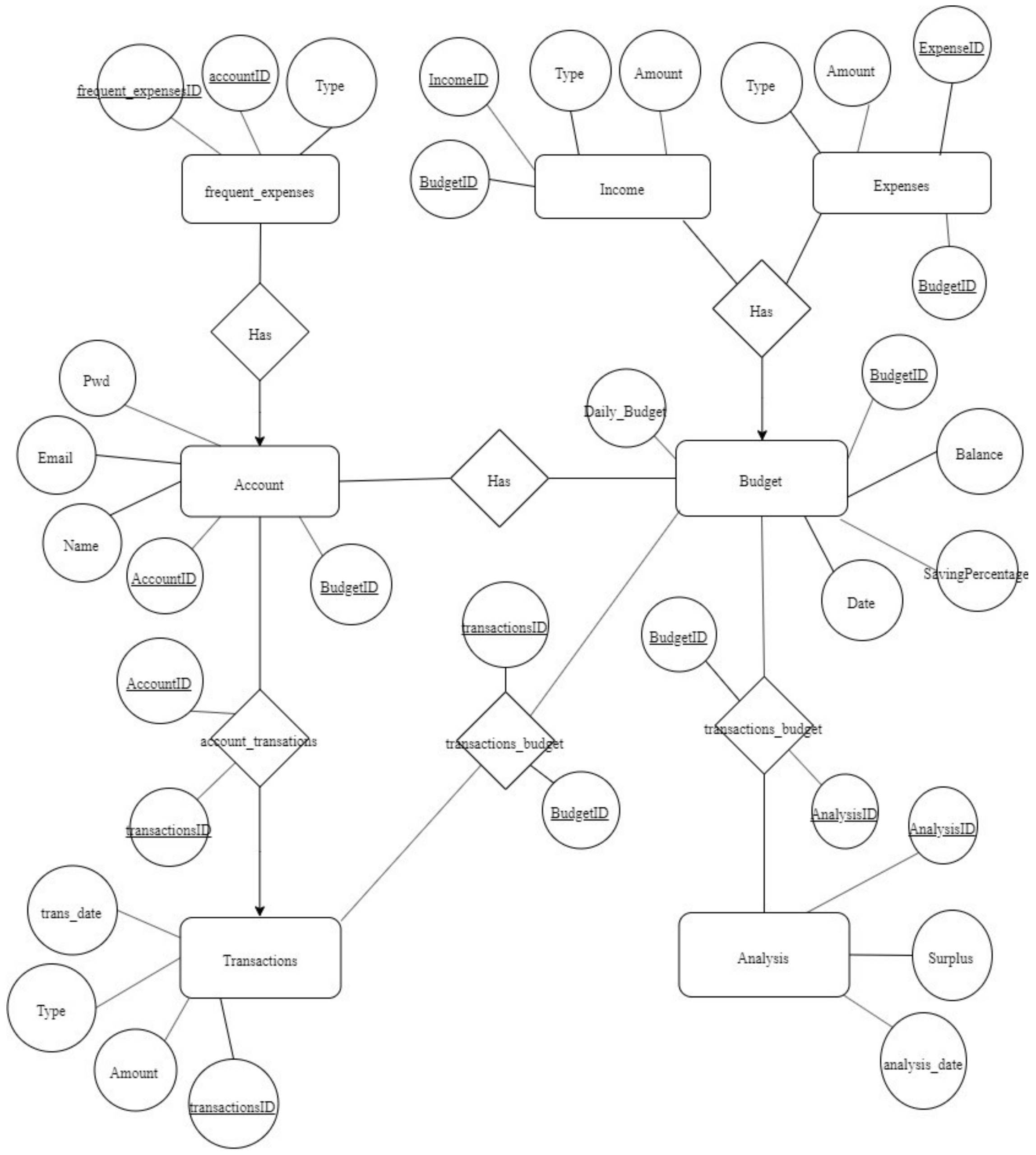


Figure 1: Entity Relationship Diagram

Schema

Accounts (accountsID, name, email, password, budgetID)
Accoount_Transactions (accounts_accountID, transactions_transactionID)
Analysis (AcctNo, Type, Overdraft, Surplus, AccumulatedSaving)
Budget (budgetID, balance, daily_budget, savingPercentage, date)
Budget_analysis (budget_budgetID, analysis_analysisID)
Income (incomeID, type, amount, budgetID)
Expenses (expenseID, type, amount, budgetID)
Frequent_Expeneses (frequent_expensesID, type, accountID)
Transactions (transactionID, type, trans_date, amount)
Transactions_budget (transactionID, budgetID)

Schema Description

Accounts: It is an entity set that describes the owner of each budget account, and has a one-to-one relationship with the budget table.

Account_Transactions: It is a relationship table between account and transaction where each account may have several transactions.

Analysis: This entity stores data that were calculated from the Budget and Accounts tables. The content of the table represents the state of the budget on whether it has a surplus amount.

Budget: This entity is the account for each user, and there are two aspects of relationships. One aspect is a one-to-one relationship with Users, and another is a one-to-many relationship with Income and Expenses.

Budget_analysis: It is a relationship table between budget and analysis tables where each budget may have several analyses.

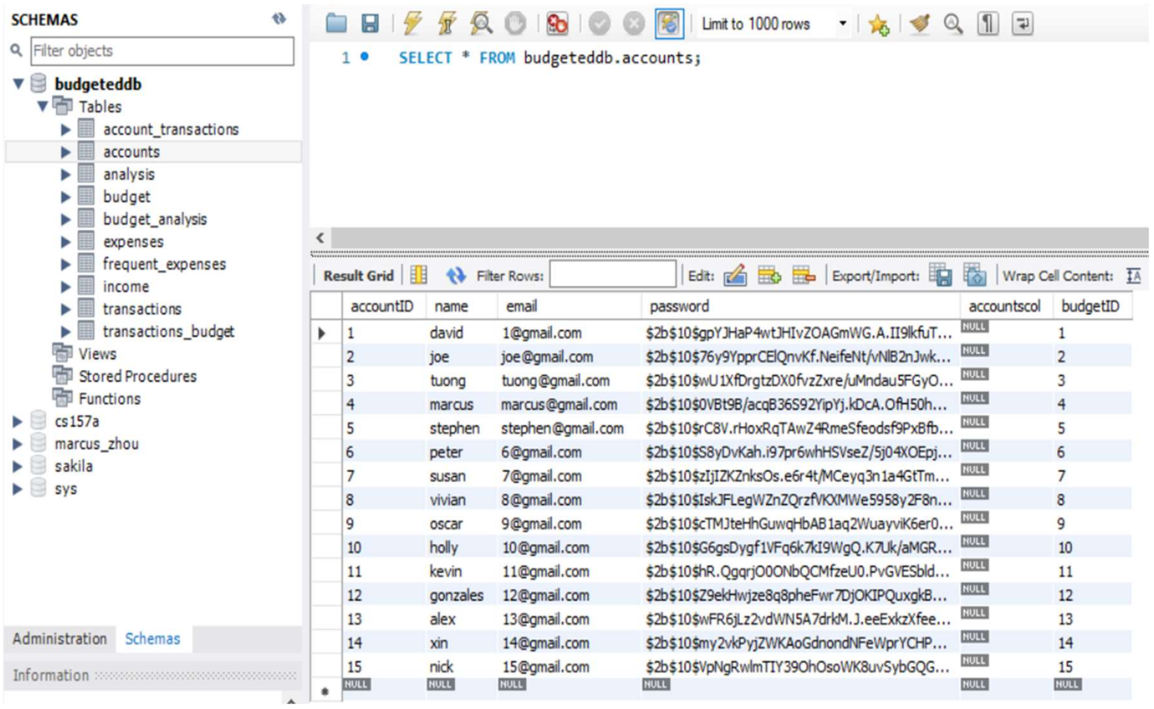
Income: Income table has a one-to-many relationship with the budget table as each budget has many transactions of recurring income like salary, bonus and so on.

Expenses: The expense table has a one-to-many relationship with the budget table as each budget has many transactions of recurring expenses like rent, insurance, and so on.

Frequent_Expenses: This table has a one-to-many relationship with the accounts table as each account has many frequent income and expenses.
Transactions: This entity set describes the details of the transaction being made and has two three-way relationships; one with Users and Budget, and one with Budget and Analysis.

Transactions_Budget: It is a relationship table between transaction and budget where all transactional data flow into the budget table.
Examples Tables

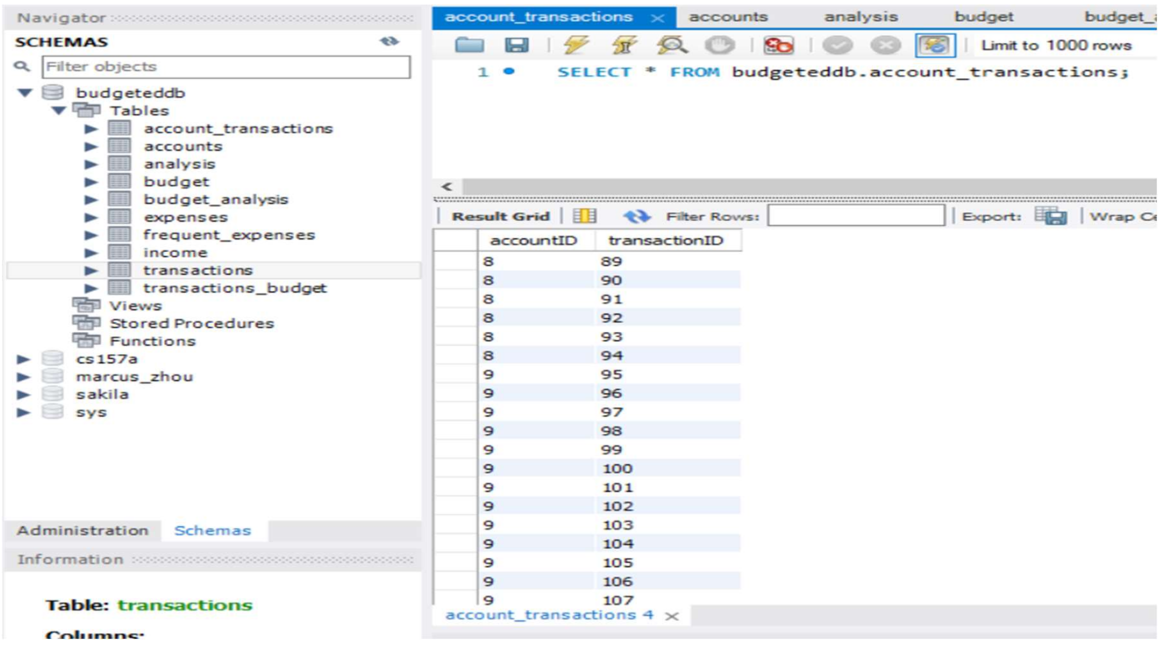
Accounts



```
SELECT * FROM budgeteddb.accounts;
```

accountID	name	email	password	accountscol	budgetID
1	david	1@gmail.com	$2b$10$gpYJHaP4wtJHIvZOAGmWG.A.II9lkfuT...	NULL	1
2	joe	joe@gmail.com	$2b$10$76y9YpprCElQnvKf.NeifeNt/vNlB2nJwk...	NULL	2
3	tuong	tuong@gmail.com	$2b$10$wU1XfDrgtzDX0fvzZxre/uMndau5FGyO...	NULL	3
4	marcus	marcus@gmail.com	$2b$10$0VBt9B/acqB36S92YipYj.kDcA.OfH50h...	NULL	4
5	stephen	stephen@gmail.com	$2b$10$rC8V.rHoxRqTAwZ4RmeSfeodsf9PxBfb...	NULL	5
6	peter	6@gmail.com	$2b$10$S8yDvKah.i97pr6whHSVseZ/5j04XOEpj...	NULL	6
7	susan	7@gmail.com	$2b$10$zIjIZKZnksOs.e6r4t/MCeyq3n1a4GtTm...	NULL	7
8	vivian	8@gmail.com	$2b$10$IskJFLegWZnZQrzfVKXMWe5958y2F8n...	NULL	8
9	oscar	9@gmail.com	$2b$10$cTMJteHhGuwqHbAB1aq2WuayviK6er0...	NULL	9
10	holly	10@gmail.com	$2b$10$G6gsDygf1VFq6k7kI9WgQ.K7Uk/aMGR...	NULL	10
11	kevin	11@gmail.com	$2b$10$hR.QgqrjO0ONbQCMfzeU0.PvGVESbld...	NULL	11
12	gonzales	12@gmail.com	$2b$10$Z9ekHwjze8q8pheFwr7DjOKIPQuxgkB...	NULL	12
13	alex	13@gmail.com	$2b$10$wFR6jLz2vdWN5A7drkM.J.eeExkzXfee...	NULL	13
14	xin	14@gmail.com	$2b$10$my2vkPyjZWKAoGdnondNFeWprYCHP...	NULL	14
15	nick	15@gmail.com	$2b$10$VpNgRwlmTIY39OhOsoWK8uvSybGQG...	NULL	15
NULL	NULL	NULL	NULL	NULL	NULL

Account_Transactions



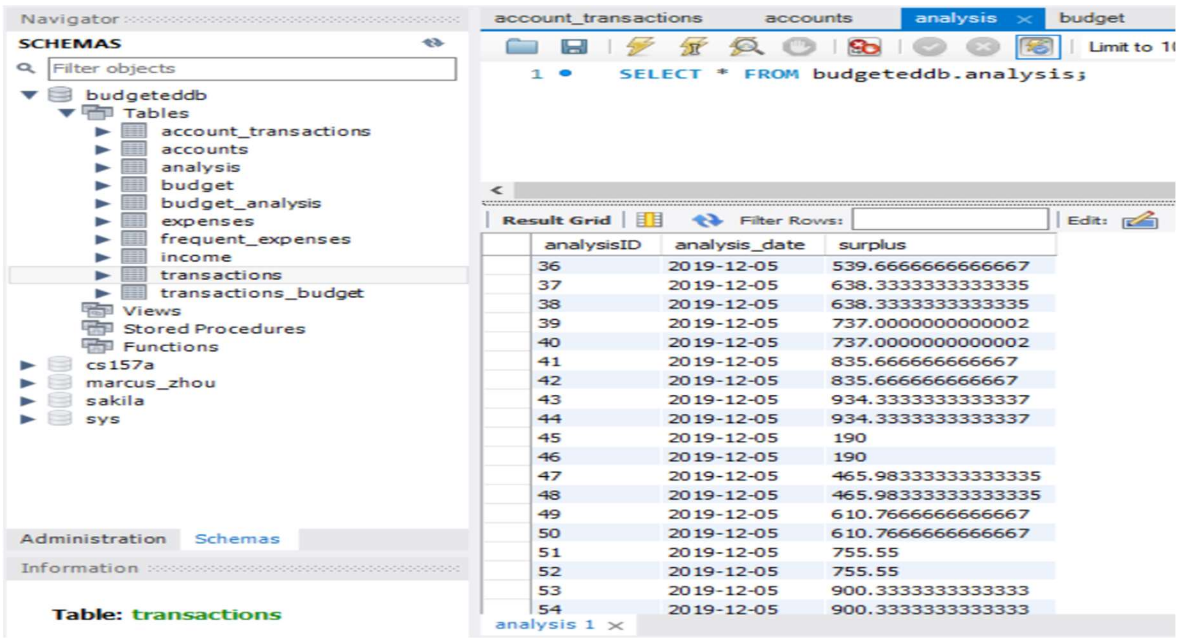
```
SELECT * FROM budgeteddb.account_transactions;
```

accountID	transactionID
8	89
8	90
8	91
8	92
8	93
8	94
9	95
9	96
9	97
9	98
9	99
9	100
9	101
9	102
9	103
9	104
9	105
9	106
9	107

Table: transactions

Columns:

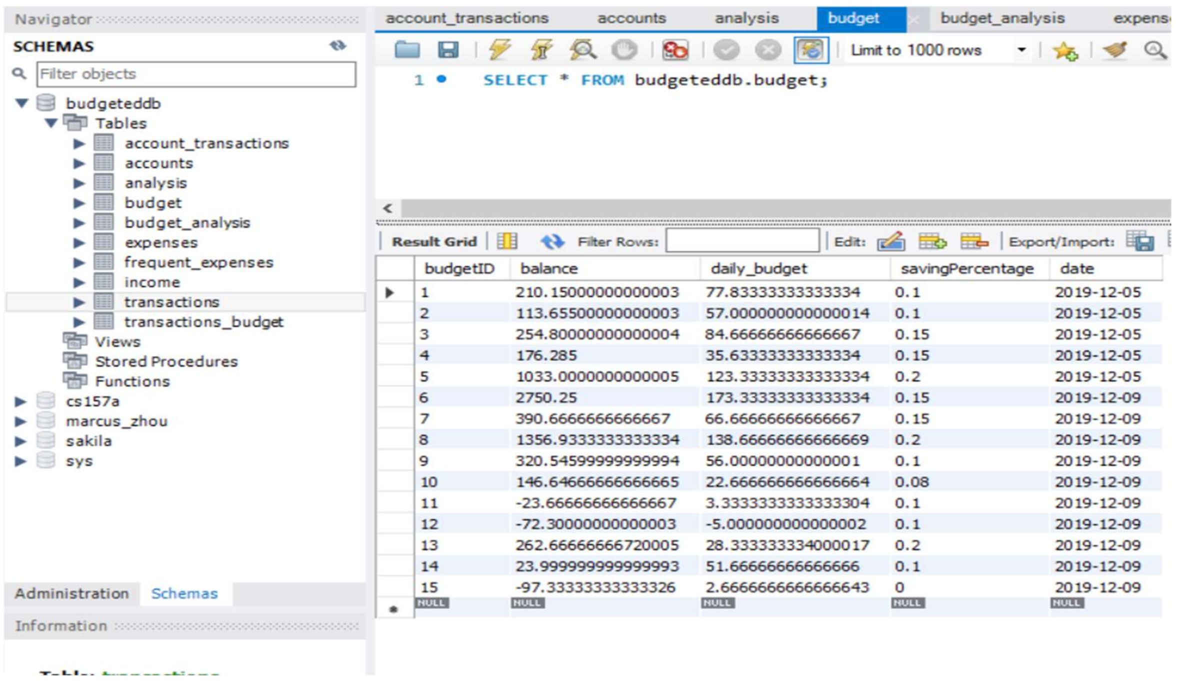
Analysis



```
SELECT * FROM budgeteddb.analysis;
```

analysisID	analysis_date	surplus
36	2019-12-05	539.6666666666667
37	2019-12-05	638.3333333333335
38	2019-12-05	638.3333333333335
39	2019-12-05	737.0000000000002
40	2019-12-05	737.0000000000002
41	2019-12-05	835.666666666667
42	2019-12-05	835.666666666667
43	2019-12-05	934.3333333333337
44	2019-12-05	934.3333333333337
45	2019-12-05	190
46	2019-12-05	190
47	2019-12-05	465.98333333333335
48	2019-12-05	465.98333333333335
49	2019-12-05	610.7666666666667
50	2019-12-05	610.7666666666667
51	2019-12-05	755.55
52	2019-12-05	755.55
53	2019-12-05	900.3333333333333
54	2019-12-05	900.3333333333333

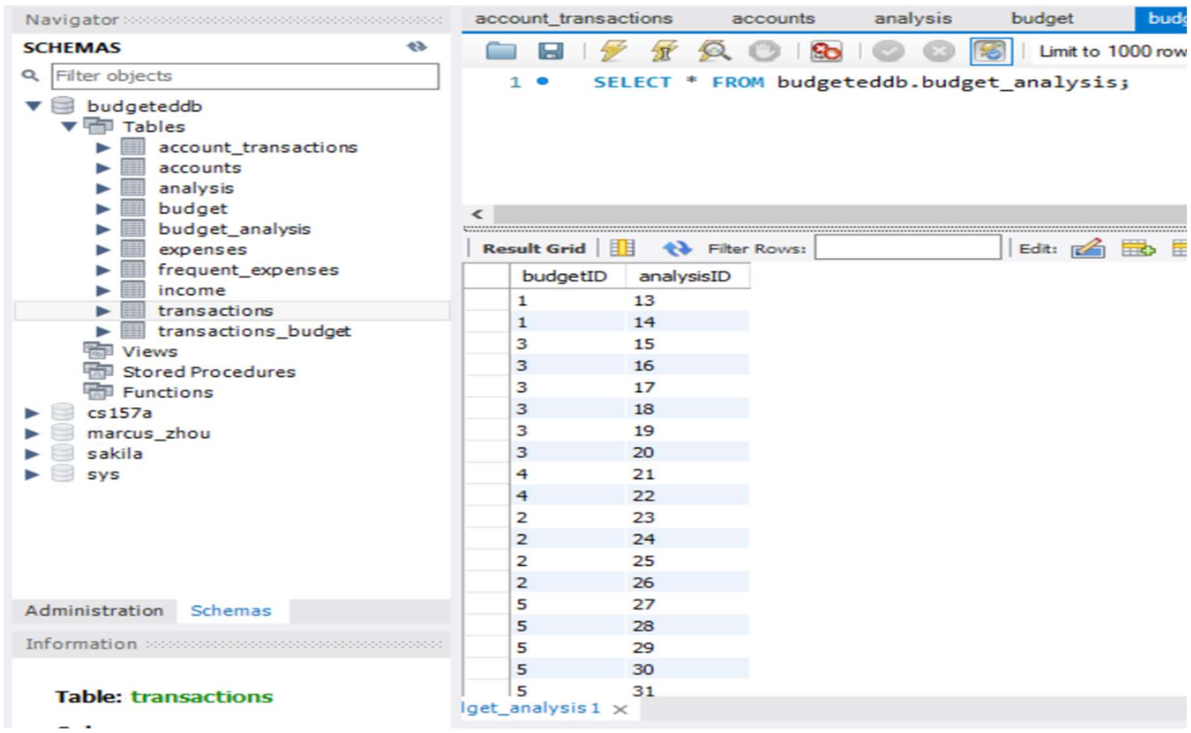
Budget



```
SELECT * FROM budgeteddb.budget;
```

budgetID	balance	daily_budget	savingPercentage	date
1	210.15000000000003	77.83333333333334	0.1	2019-12-05
2	113.65500000000003	57.000000000000014	0.1	2019-12-05
3	254.80000000000004	84.66666666666667	0.15	2019-12-05
4	176.285	35.63333333333334	0.15	2019-12-05
5	1033.0000000000005	123.33333333333334	0.2	2019-12-05
6	2750.25	173.33333333333334	0.15	2019-12-09
7	390.6666666666667	66.66666666666667	0.15	2019-12-09
8	1356.9333333333334	138.66666666666669	0.2	2019-12-09
9	320.54599999999994	56.00000000000001	0.1	2019-12-09
10	146.64666666666665	22.666666666666664	0.08	2019-12-09
11	-23.666666666666667	3.3333333333333304	0.1	2019-12-09
12	-72.30000000000003	-5.000000000000002	0.1	2019-12-09
13	262.66666666720005	28.333333334000017	0.2	2019-12-09
14	23.999999999999993	51.66666666666666	0.1	2019-12-09
15	-97.33333333333326	2.6666666666666643	0	2019-12-09
NULL	NULL	NULL	NULL	NULL

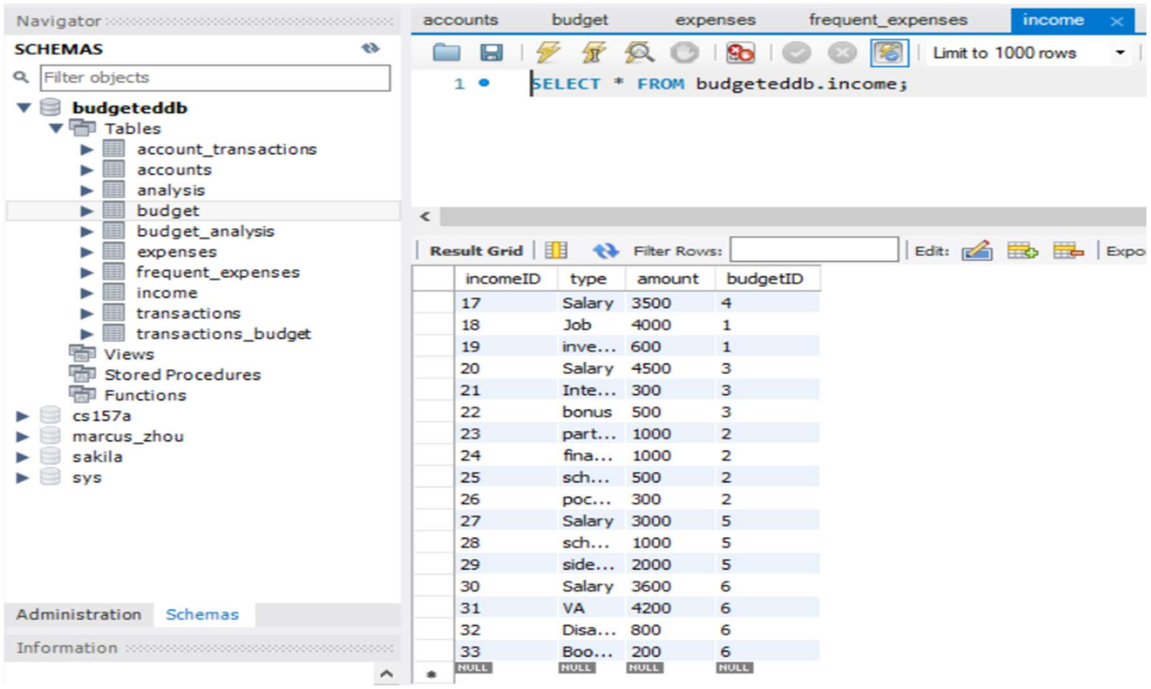
Budget_Analysis



```
SELECT * FROM budgeteddb.budget_analysis;
```

budgetID	analysisID
1	13
1	14
3	15
3	16
3	17
3	18
3	19
3	20
4	21
4	22
2	23
2	24
2	25
2	26
5	27
5	28
5	29
5	30
5	31

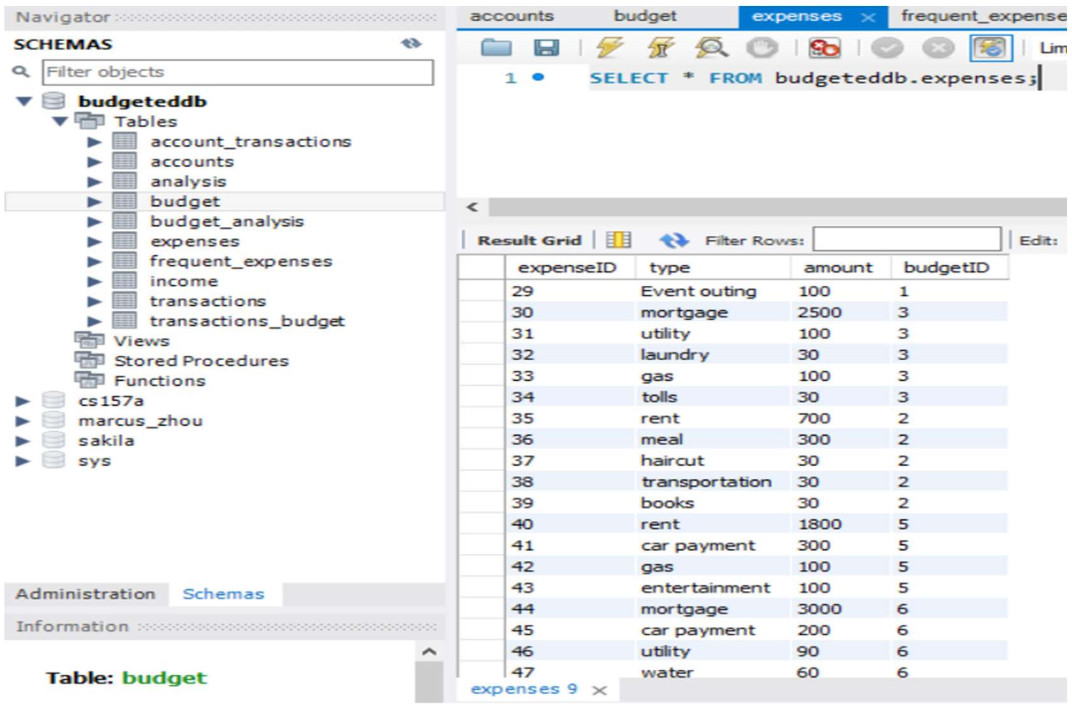
Income



```
SELECT * FROM budgeteddb.income;
```

incomeID	type	amount	budgetID
17	Salary	3500	4
18	Job	4000	1
19	inve...	600	1
20	Salary	4500	3
21	Inte...	300	3
22	bonus	500	3
23	part...	1000	2
24	fina...	1000	2
25	sch...	500	2
26	poc...	300	2
27	Salary	3000	5
28	sch...	1000	5
29	side...	2000	5
30	Salary	3600	6
31	VA	4200	6
32	Disa...	800	6
33	Boo...	200	6
NULL	NULL	NULL	NULL

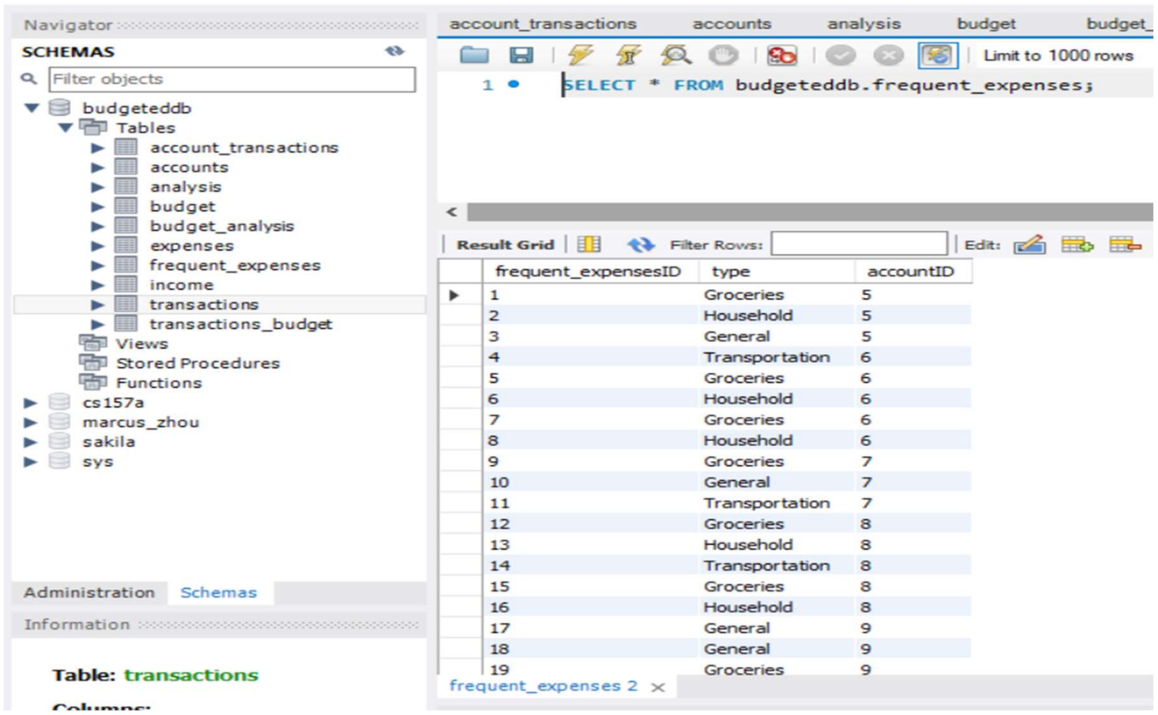
Expenses



```
SELECT * FROM budgeteddb.expenses;
```

expenseID	type	amount	budgetID
29	Event outing	100	1
30	mortgage	2500	3
31	utility	100	3
32	laundry	30	3
33	gas	100	3
34	tolls	30	3
35	rent	700	2
36	meal	300	2
37	haircut	30	2
38	transportation	30	2
39	books	30	2
40	rent	1800	5
41	car payment	300	5
42	gas	100	5
43	entertainment	100	5
44	mortgage	3000	6
45	car payment	200	6
46	utility	90	6
47	water	60	6

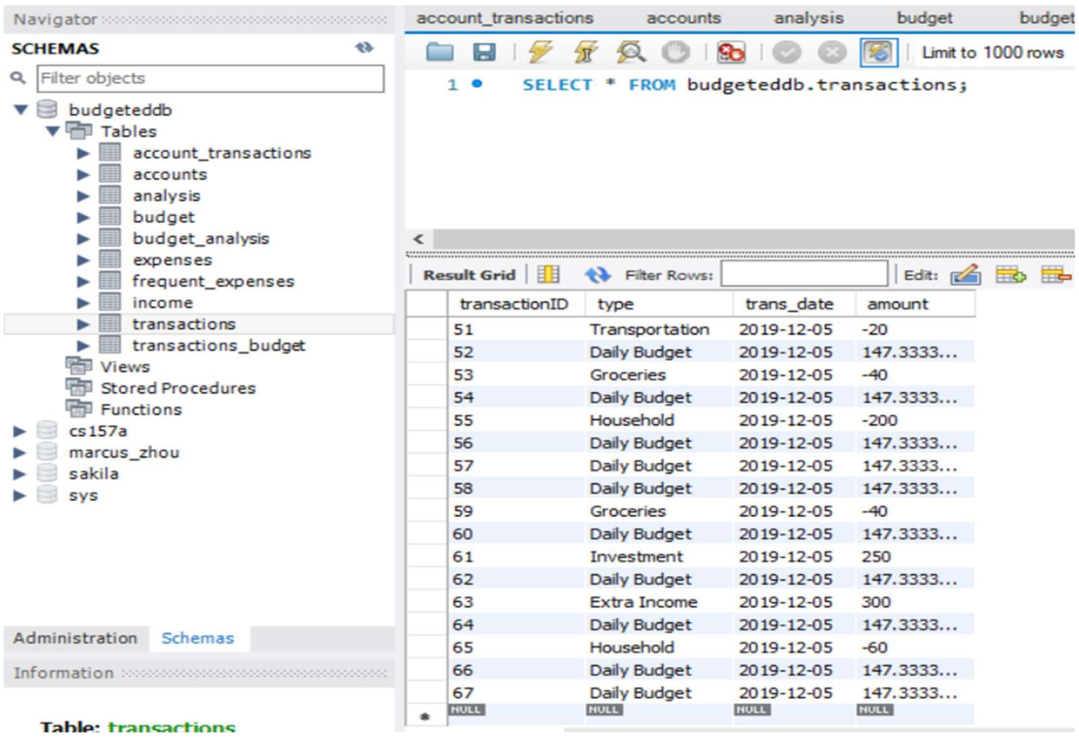
Frequent_Expenses



```
SELECT * FROM budgeteddb.frequent_expenses;
```

frequent_expensesID	type	accountID
1	Groceries	5
2	Household	5
3	General	5
4	Transportation	6
5	Groceries	6
6	Household	6
7	Groceries	6
8	Household	6
9	Groceries	7
10	General	7
11	Transportation	7
12	Groceries	8
13	Household	8
14	Transportation	8
15	Groceries	8
16	Household	8
17	General	9
18	General	9
19	Groceries	9

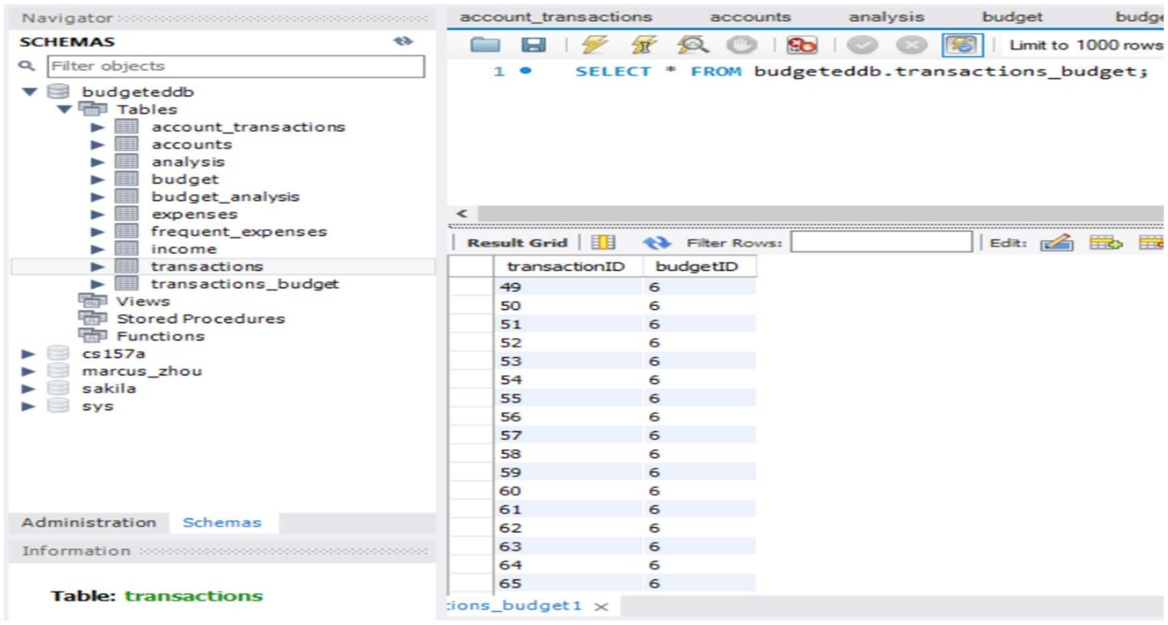
Transactions



```sql
SELECT * FROM budgeteddb.transactions;
```

transactionID	type	trans_date	amount
51	Transportation	2019-12-05	-20
52	Daily Budget	2019-12-05	147.3333...
53	Groceries	2019-12-05	-40
54	Daily Budget	2019-12-05	147.3333...
55	Household	2019-12-05	-200
56	Daily Budget	2019-12-05	147.3333...
57	Daily Budget	2019-12-05	147.3333...
58	Daily Budget	2019-12-05	147.3333...
59	Groceries	2019-12-05	-40
60	Daily Budget	2019-12-05	147.3333...
61	Investment	2019-12-05	250
62	Daily Budget	2019-12-05	147.3333...
63	Extra Income	2019-12-05	300
64	Daily Budget	2019-12-05	147.3333...
65	Household	2019-12-05	-60
66	Daily Budget	2019-12-05	147.3333...
67	Daily Budget	2019-12-05	147.3333...
NULL	NULL	NULL	NULL

Transaction_Budget



```sql
SELECT * FROM budgeteddb.transactions_budget;
```

transactionID	budgetID
49	6
50	6
51	6
52	6
53	6
54	6
55	6
56	6
57	6
58	6
59	6
60	6
61	6
62	6
63	6
64	6
65	6

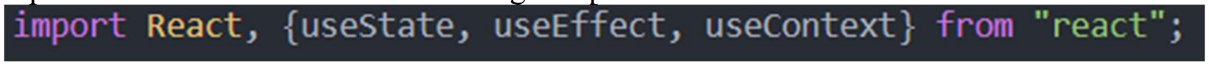
Implementation

Description

Our application follows 3-tier architecture, a frontend client for data input, server for operational computations, and MySQL database for data storage. During the implementation, we utilize JavaScript, HTML, and CSS languages for both backend and frontend using frameworks; ReactJS, NodeJS, and ExpressJS while using MySQL to retrieve data from the database, MySQL Workbench 8.0. The files are stored as "JSON" files to build the frontend web pages where users enter data and the backend server where all the routing and navigation happen based on the command entered on the webpages.

Design Implementation

Before webpages, we implemented "react-router-dom" package on the server that contains "BrowserRouter.js", "router.js" and "Link.js" to navigate between webpages. Once the object "Link.js" was built, the object is imported into each webpage component that enables links on the texts to act as a button for navigating to the appropriate webpages. We also use "Hooks" to represent the state of the function among components.

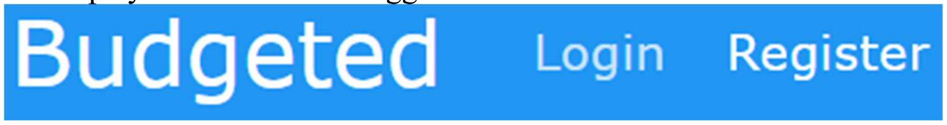


```
import React, {useState, useEffect, useContext} from "react";
```

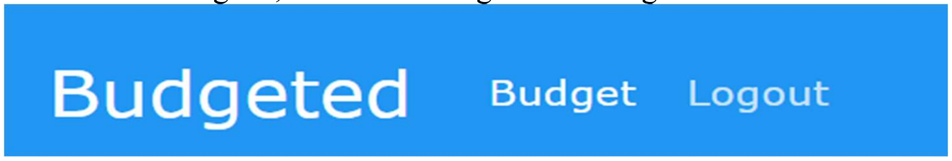
A total of 11 components was implemented to build the frontend webpages.

Budgeted
Before user login, this component acts as a link to the homepage where the list of registered users is displayed. After login, when the user clicks on the Budgeted, it'll take the user to the dashboard where the individual budget is displayed in an animation.

Nav
Nav component is a fixed component resided on the top of the webpage where there are links to home, registration, and login pages. Besides the "Budgeted" button, login and registration links are displayed if no user has logged in.

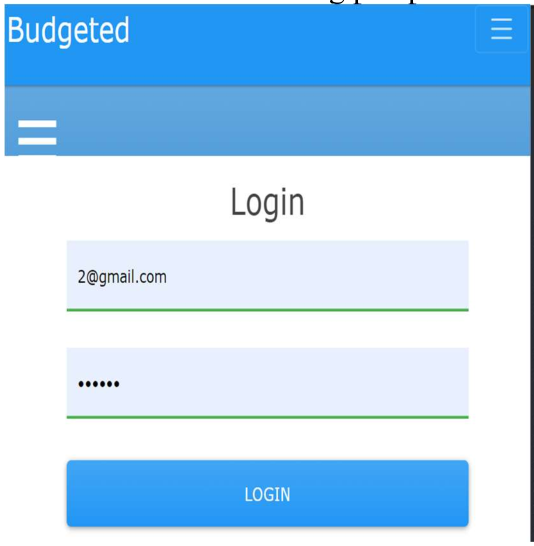


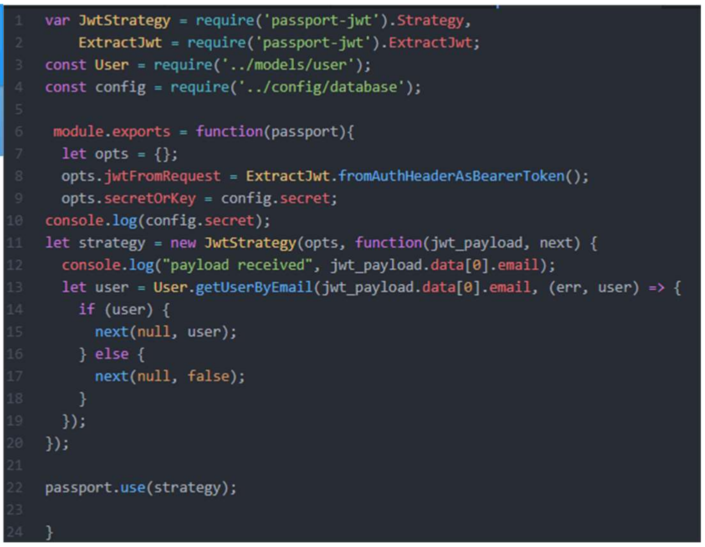
After the user logs in, there are "Budget" and "Log out" button.



Login
This component is for registered users to enter the system by entering their email addresses and user password. We implemented middleware for our authentication process in which we use

Passport.js and JSON web tokens. The password entries are matched with hashed password in the backend database using passport.

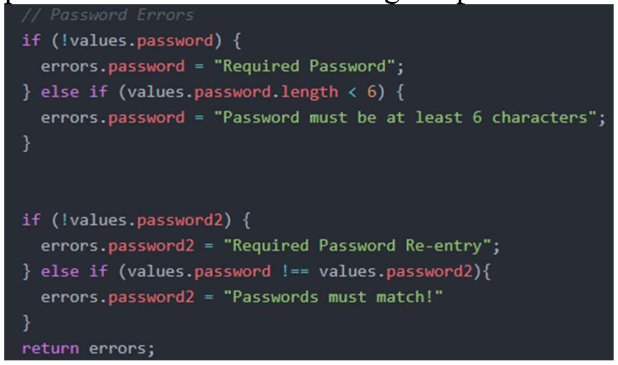




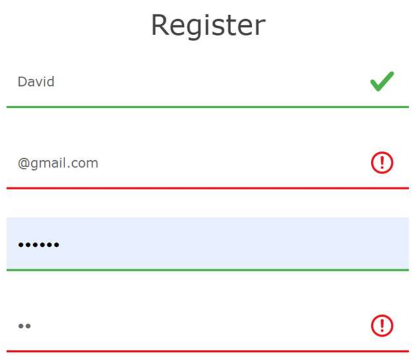
```
var JwtStrategy = require('passport-jwt').Strategy,
    ExtractJwt = require('passport-jwt').ExtractJwt;
const User = require('../models/user');
const config = require('../config/database');

 module.exports = function(passport){
  let opts = {};
  opts.jwtFromRequest = ExtractJwt.fromAuthHeaderAsBearerToken();
  opts.secretOrKey = config.secret;
console.log(config.secret);
let strategy = new JwtStrategy(opts, function(jwt_payload, next) {
  console.log("payload received", jwt_payload.data[0].email);
  let user = User.getUserByEmail(jwt_payload.data[0].email, (err, user) => {
    if (user) {
      next(null, user);
    } else {
      next(null, false);
    }
  });
});

passport.use(strategy);

}
```

Register

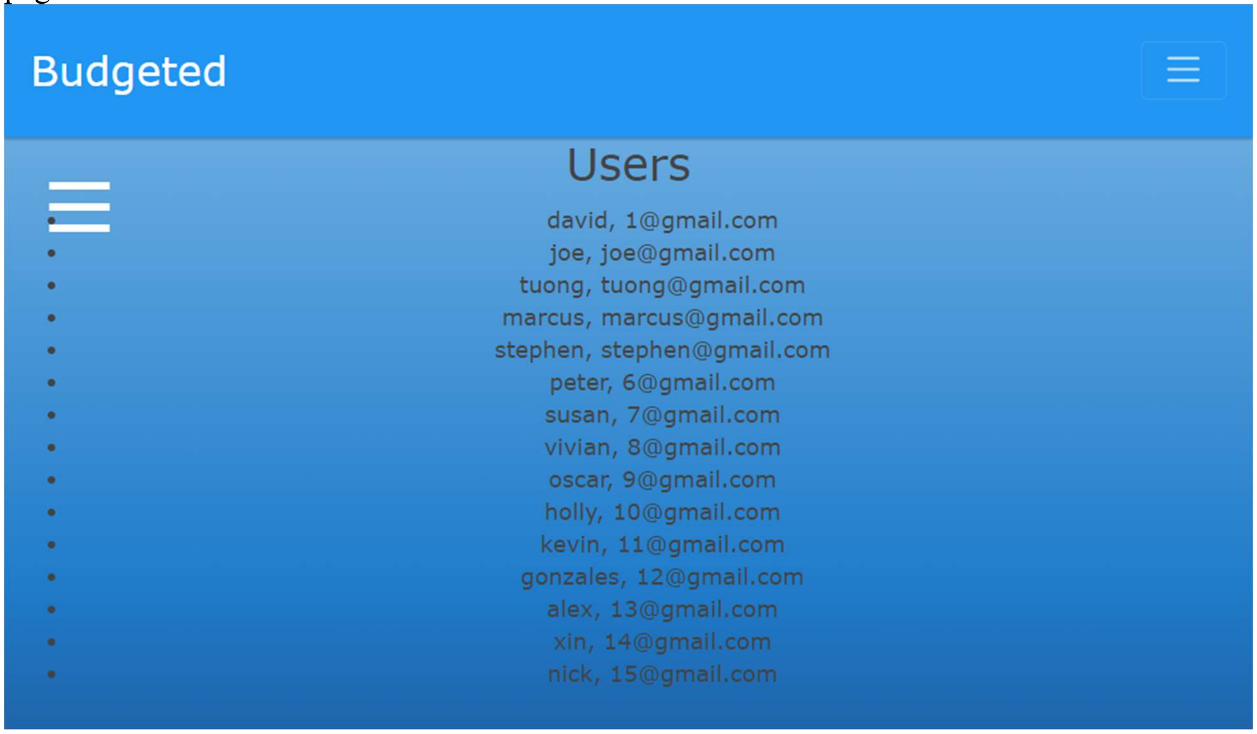
Register component uses form validation to verify the correctness of the data entered by users. For instance, users are required to enter a valid email address. They are also required to enter the password that matches the original password.



```
// Password Errors
if (!values.password) {
  errors.password = "Required Password";
} else if (values.password.length < 6) {
  errors.password = "Password must be at least 6 characters";
}


if (!values.password2) {
  errors.password2 = "Required Password Re-entry";
} else if (values.password !== values.password2){
  errors.password2 = "Passwords must match!"
}
return errors;
```



Users
Users component displays all users registered and stored in the database system on the home page.

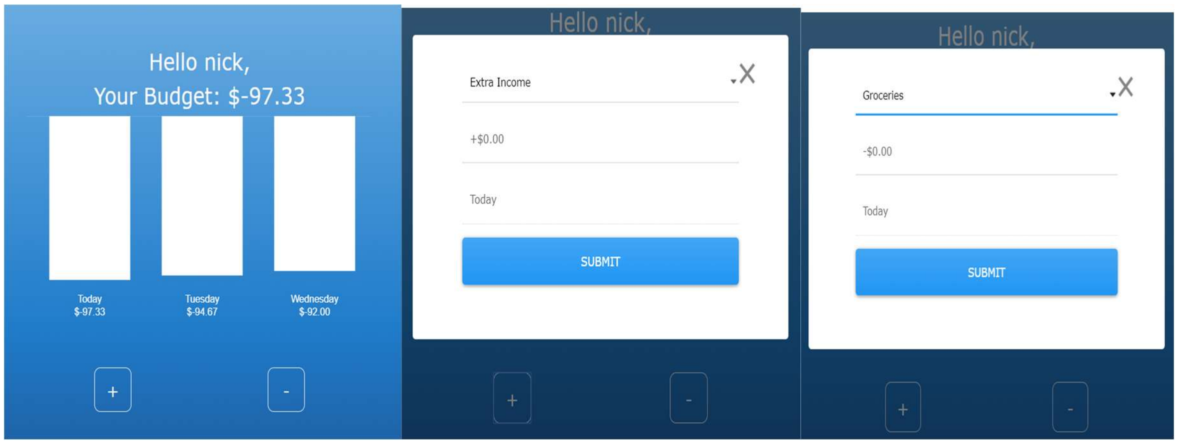


Budgeted

Users

- david, 1@gmail.com
- joe, joe@gmail.com
- tuong, tuong@gmail.com
- marcus, marcus@gmail.com
- stephen, stephen@gmail.com
- peter, 6@gmail.com
- susan, 7@gmail.com
- vivian, 8@gmail.com
- oscar, 9@gmail.com
- holly, 10@gmail.com
- kevin, 11@gmail.com
- gonzales, 12@gmail.com
- alex, 13@gmail.com
- xin, 14@gmail.com
- nick, 15@gmail.com

Dashboard
The dashboard component displays data in an animation graph based on the current day's budget. It also forecasts the budget for the next coming two days using today's budget. From the Dashboard, users will be able to add their unplanned income and expenses for the current day.



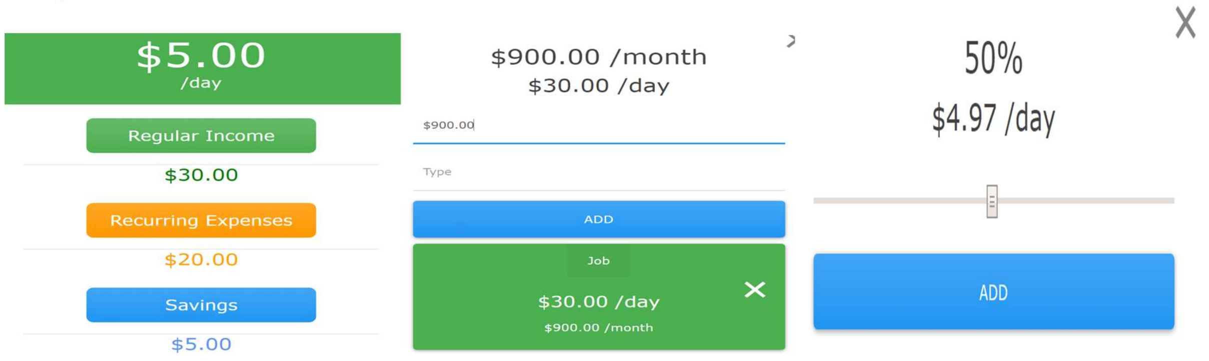
Hello nick,
Your Budget: $-97.33

Today $-97.33
Tuesday $-94.67
Wednesday $-92.00

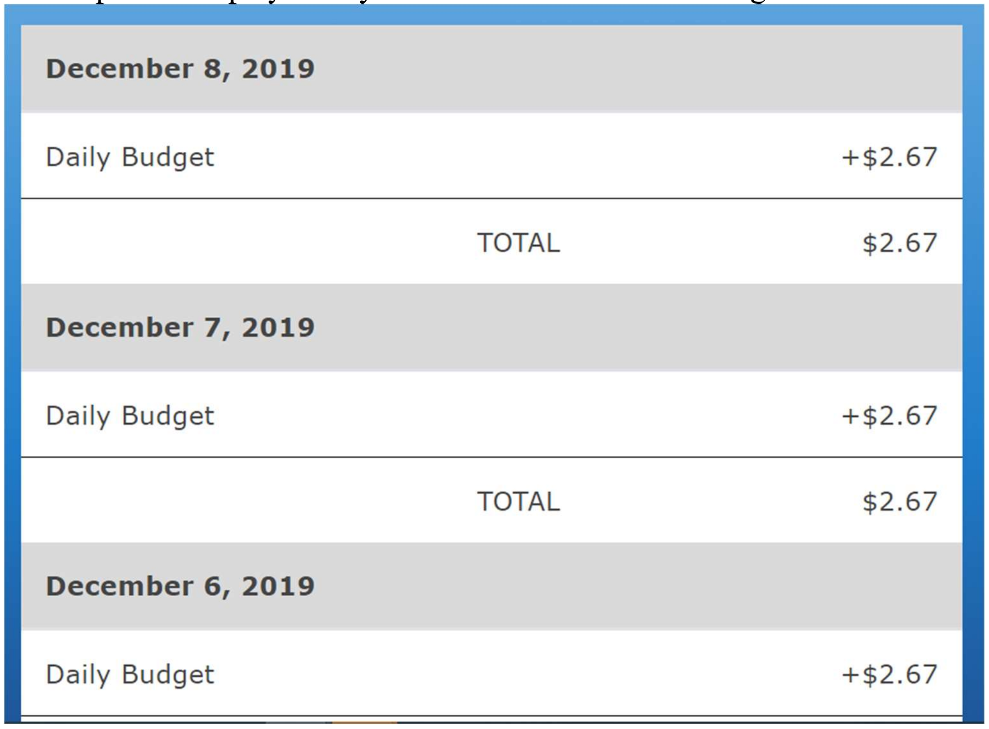
Hello nick,
Extra Income
+$0.00
Today

Hello nick,
Groceries
-$0.00
Today

Daily Budget
Daily Budget allows users to enter data for income, recurring expense and saving amount. The total income subtracts recurring expenses and saving, the user will know his daily budget. For example, the user can enter the monthly amount for regular income that calculates the daily income. Similarly, the user can enter recurring expenses per month that calculate the daily expenses. On top of income and expense, the user can also enter the percentage of the daily budget that the individual wants to save.



Transactions
Transactions component displays today's date with the available budget.



December 8, 2019

Daily Budget	+$2.67
TOTAL	$2.67

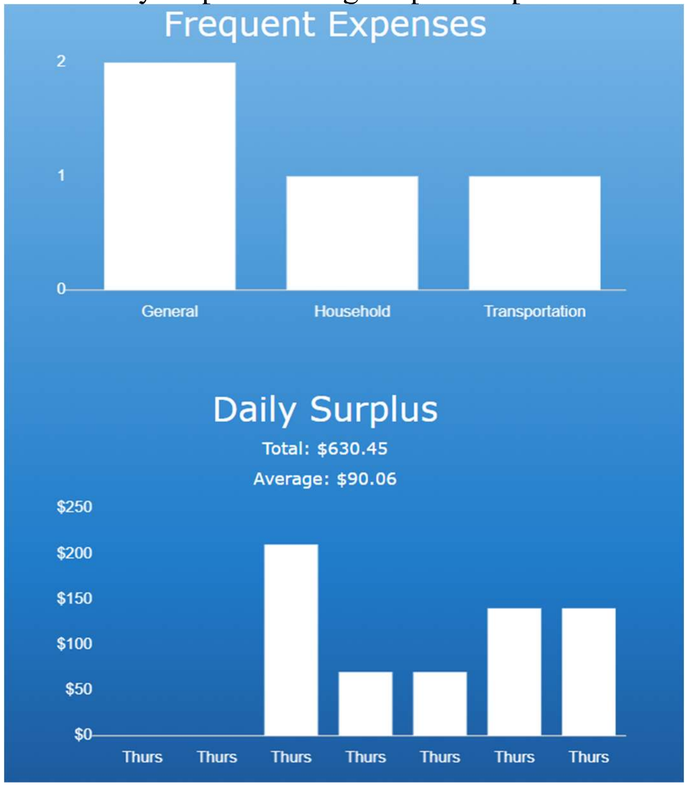
December 7, 2019

Daily Budget	+$2.67
TOTAL	$2.67

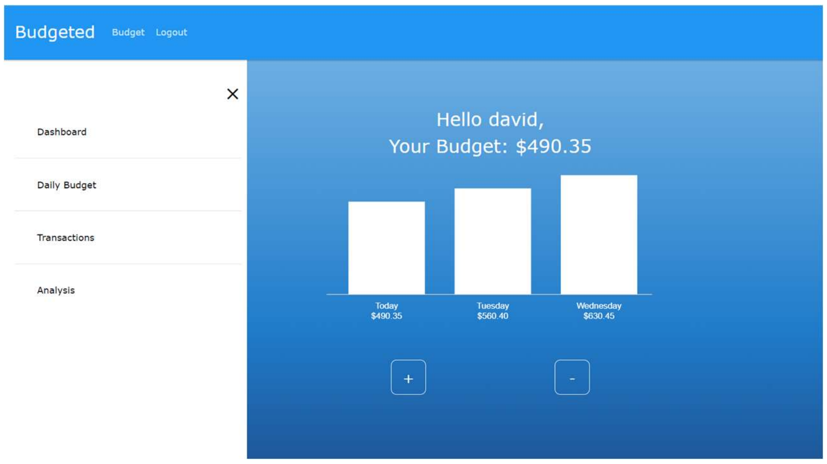
December 6, 2019

Daily Budget	+$2.67

Analysis
The analysis page calculates daily surpluses using frequent expenses and recurring budget.



Sidebar
The sidebar is similar to the Nav component where users can navigate through different pages. It was implemented with "Link.js" attaching to each text.



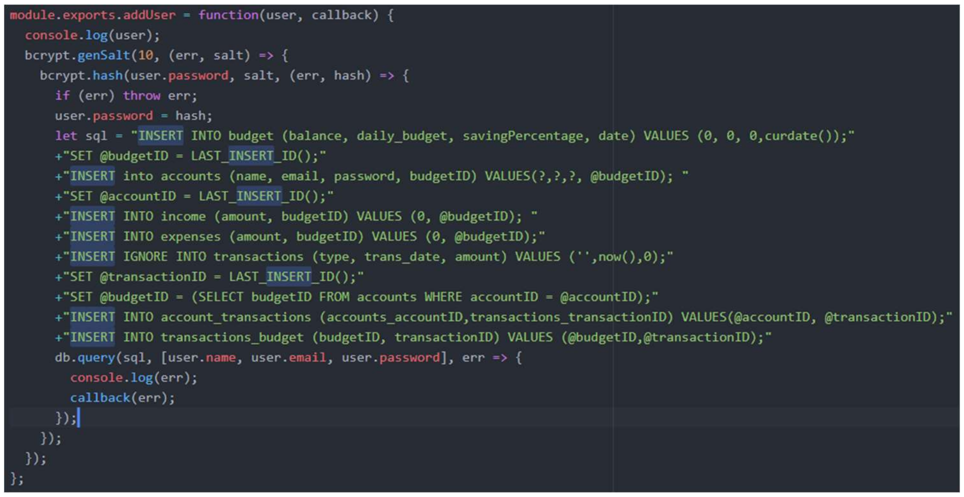
Toggle

Toggle is the button to open the sidebar. It was implemented with CSS to insert the animation when being hovered. Three horizontal bar button becomes vertical when the mouse hovers onto the button.

Insert, Delete, Update, and Query Examples

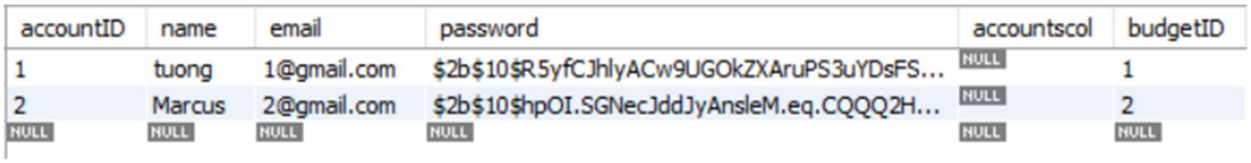
Insert

Registration inserts new user information to the database. As well as insert into relationship tables and instantiate a transaction for that user.



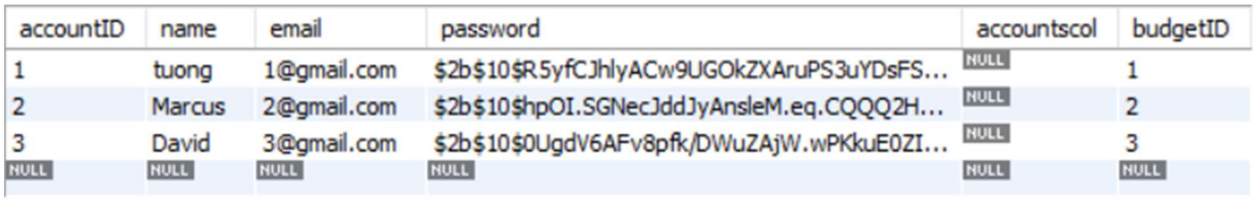
```
module.exports.addUser = function(user, callback) {
  console.log(user);
  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(user.password, salt, (err, hash) => {
      if (err) throw err;
      user.password = hash;
      let sql = "INSERT INTO budget (balance, daily_budget, savingPercentage, date) VALUES (0, 0, 0,curdate());"
      +"SET @budgetID = LAST_INSERT_ID();"
      +"INSERT into accounts (name, email, password, budgetID) VALUES(?,?,?, @budgetID); "
      +"SET @accountID = LAST_INSERT_ID();"
      +"INSERT INTO income (amount, budgetID) VALUES (0, @budgetID); "
      +"INSERT INTO expenses (amount, budgetID) VALUES (0, @budgetID);"
      +"INSERT IGNORE INTO transactions (type, trans_date, amount) VALUES ('',now(),0);"
      +"SET @transactionID = LAST_INSERT_ID();"
      +"SET @budgetID = (SELECT budgetID FROM accounts WHERE accountID = @accountID);"
      +"INSERT INTO account_transactions (accounts_accountID,transactions_transactionID) VALUES(@accountID, @transactionID);"
      +"INSERT INTO transactions_budget (budgetID, transactionID) VALUES (@budgetID,@transactionID);"
      db.query(sql, [user.name, user.email, user.password], err => {
        console.log(err);
        callback(err);
      });
    });
  });
};
```

Before:



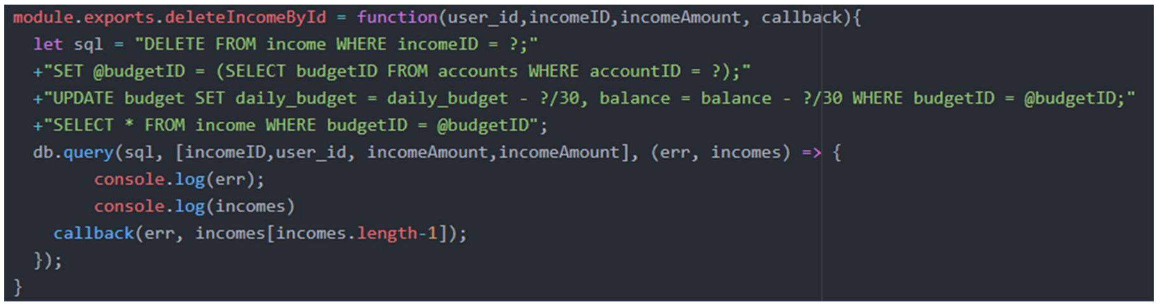
accountID	name	email	password	accountscol	budgetID
1	tuong	1@gmail.com	$2b$10$R5yfCJhlyACw9UGOkZXAruPS3uYDsFS...	NULL	1
2	Marcus	2@gmail.com	$2b$10$hpOI.SGNecJddJyAnsleM.eq.CQQQ2H...	NULL	2
NULL	NULL	NULL	NULL	NULL	NULL

After:



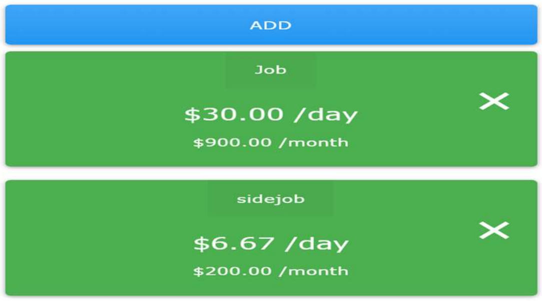
accountID	name	email	password	accountscol	budgetID
1	tuong	1@gmail.com	$2b$10$R5yfCJhlyACw9UGOkZXAruPS3uYDsFS...	NULL	1
2	Marcus	2@gmail.com	$2b$10$hpOI.SGNecJddJyAnsleM.eq.CQQQ2H...	NULL	2
3	David	3@gmail.com	$2b$10$0UgdV6AFv8pfk/DWuZAjW.wPKkuE0ZI...	NULL	3
NULL	NULL	NULL	NULL	NULL	NULL

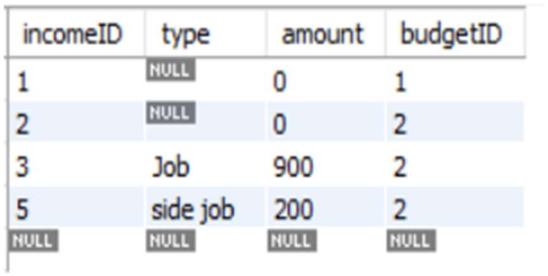
Delete
Users can also delete data entry. For example, an individual can delete income entry that was entered before.



```
module.exports.deleteIncomeById = function(user_id,incomeID,incomeAmount, callback){
  let sql = "DELETE FROM income WHERE incomeID = ?;"
  +"SET @budgetID = (SELECT budgetID FROM accounts WHERE accountID = ?);"
  +"UPDATE budget SET daily_budget = daily_budget - ?/30, balance = balance - ?/30 WHERE budgetID = @budgetID;"
  +"SELECT * FROM income WHERE budgetID = @budgetID";
  db.query(sql, [incomeID,user_id, incomeAmount,incomeAmount], (err, incomes) => {
        console.log(err);
        console.log(incomes)
    callback(err, incomes[incomes.length-1]);
  });
}
```

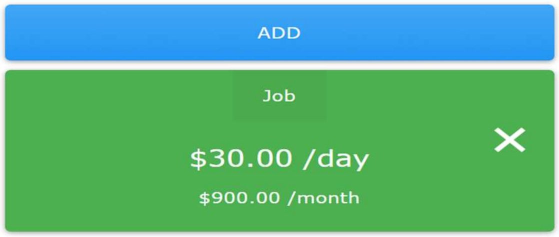
Before:

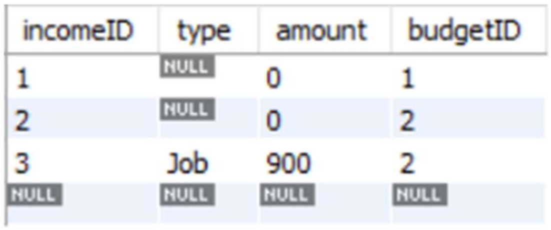




incomeID	type	amount	budgetID
1	NULL	0	1
2	NULL	0	2
3	Job	900	2
5	side job	200	2
NULL	NULL	NULL	NULL

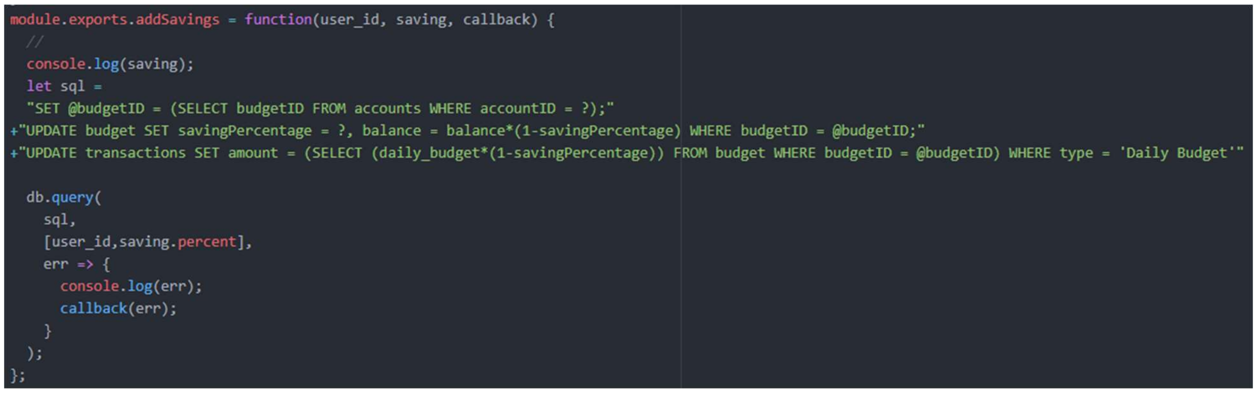
After:





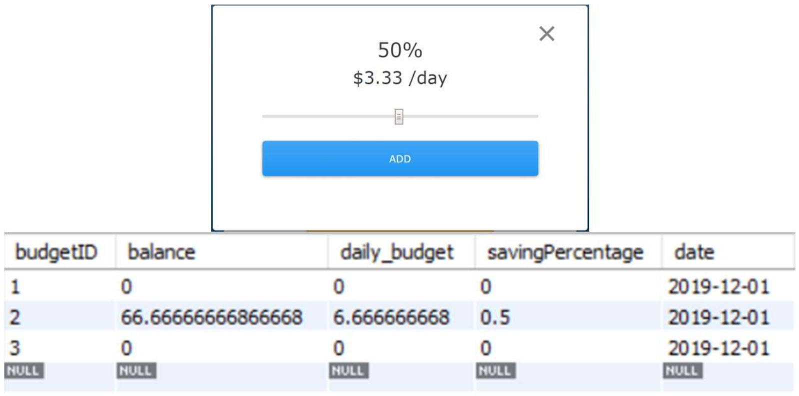
incomeID	type	amount	budgetID
1	NULL	0	1
2	NULL	0	2
3	Job	900	2
NULL	NULL	NULL	NULL

Update
Users can also change the percentage they want to save daily. In the backend, we are updating the saving percentage. For example, the user can decrease the saving percentage from 50% to 25%. In the screenshot below the budget table's balance is updated when the savings % is updated.



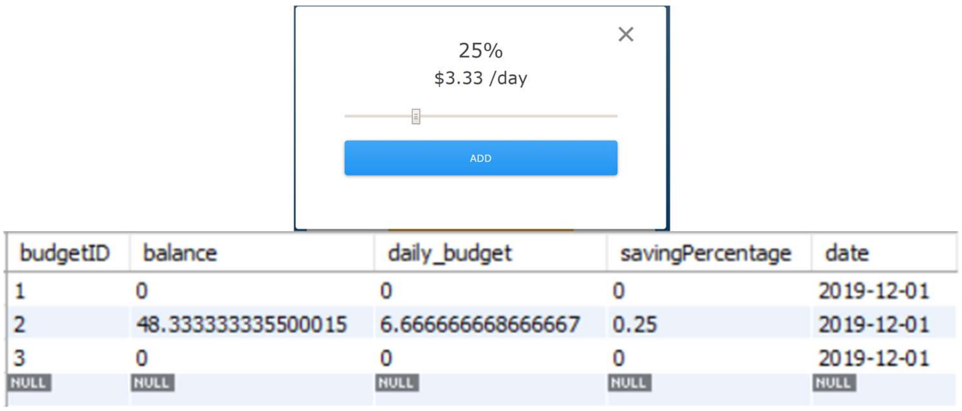
```
module.exports.addSavings = function(user_id, saving, callback) {
  //
  console.log(saving);
  let sql =
  "SET @budgetID = (SELECT budgetID FROM accounts WHERE accountID = ?);"
+"UPDATE budget SET savingPercentage = ?, balance = balance*(1-savingPercentage) WHERE budgetID = @budgetID;"
+"UPDATE transactions SET amount = (SELECT (daily_budget*(1-savingPercentage)) FROM budget WHERE budgetID = @budgetID) WHERE type = 'Daily Budget'"

  db.query(
    sql,
    [user_id,saving.percent],
    err => {
      console.log(err);
      callback(err);
    }
  );
};
```

Before:



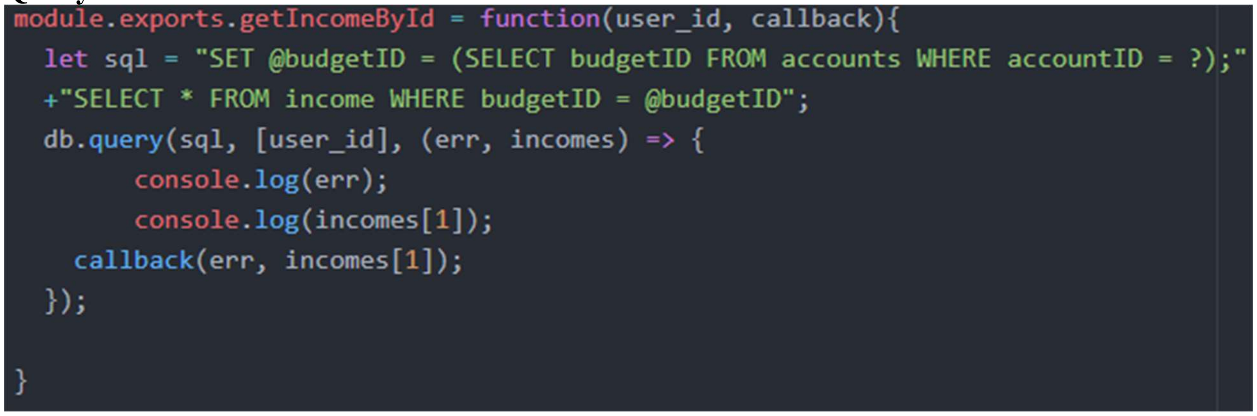
budgetID	balance	daily_budget	savingPercentage	date
1	0	0	0	2019-12-01
2	66.66666666866668	6.666666668	0.5	2019-12-01
3	0	0	0	2019-12-01
NULL	NULL	NULL	NULL	NULL

After:

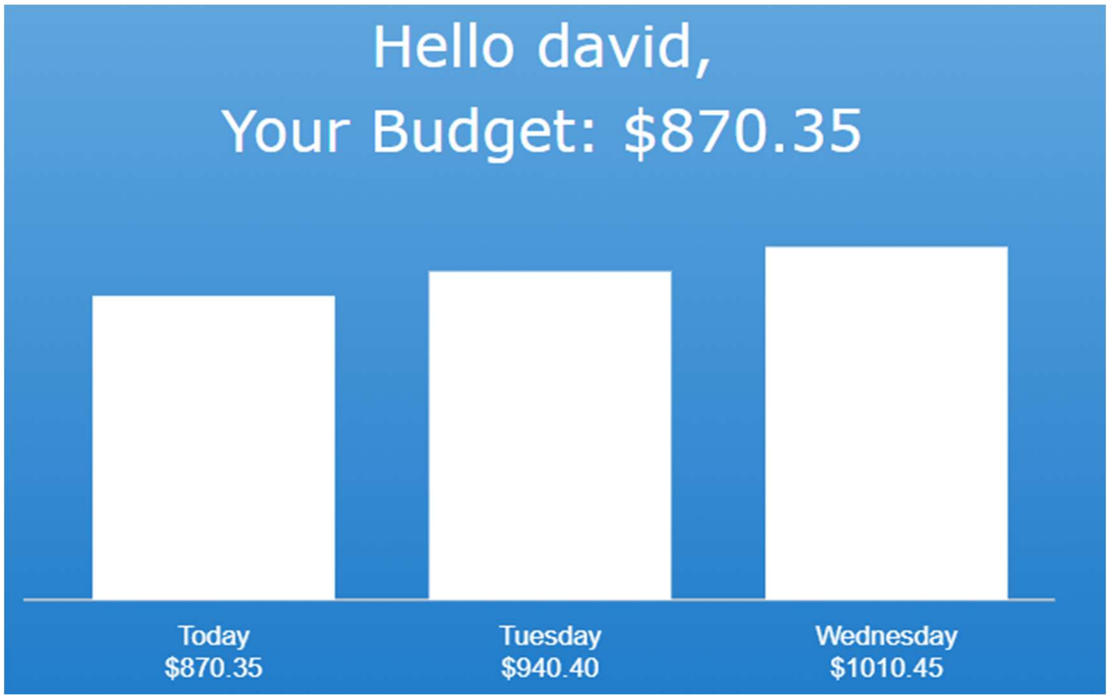


budgetID	balance	daily_budget	savingPercentage	date
1	0	0	0	2019-12-01
2	48.333333335500015	6.666666668666667	0.25	2019-12-01
3	0	0	0	2019-12-01
NULL	NULL	NULL	NULL	NULL

Query



```
module.exports.getIncomeById = function(user_id, callback){
  let sql = "SET @budgetID = (SELECT budgetID FROM accounts WHERE accountID = ?);"
  +"SELECT * FROM income WHERE budgetID = @budgetID";
  db.query(sql, [user_id], (err, incomes) => {
        console.log(err);
        console.log(incomes[1]);
    callback(err, incomes[1]);
  });

}
```


Hello david,

Your Budget: $870.35

Lesson Learned

Tuong Nguyen:
By doing this project, I have learned the process of front-to-back development of a 3-tier database application. I have taken part in the design, documentation, implementation of the project. From the design aspect, I have learned that design needs several iterations throughout the project as implementation demands different design alternatives. From the front end aspect, I have learned to send and receive HTTP requests as well as using a modern front end javascript framework such as ReactJS. I have learned form validation as well as component-based web development and managing the application's state based on those components. Other front end things I've learned is to use open source libraries to achieve the project requirements. An example is I had to learn and read documentation from a library called ChartJS to display the graphs for the project. From the backend aspect, I have learned to handle HTTP requests as well as using third-party tools such as Postman for API testing. I also learned how to design and implement relational databases using concepts from the lectures. I learned to perform CRUD operations from front to back using SQL programming and making complex queries between related entities. Outside of the technical lessons, I have learned to push past my limits and learned to persevere and developed a mindset that is as long as I put in the hours, anything can be accomplished.

Zehua Liu:

Marcus Zhou:
This is the first application that I get to interact with web-based languages. Along the way, I learned HTML, CSS, Javascript as I implemented a sidebar where users can click on each link to go to other webpages. In Javascript, I learned how to use Hooks in order to make the sidebar operate opening and closing by using the "usestate" variable. I learn how to work with a database from drawing entity relation diagrams to retrieving data. I learned how to implement database schema in MySQL workbench. Editing the database scheme and design repeatedly make me aware of the importance of the iterative process of improving the project. Finally, I appreciate my reliable teammates who devote a lot of work hours to this project working with schedule constraints.

Instructions for Setting up:

Tools Required: Git, MySQL Workbench 8.0, Atom, Nodejs

Assuming all the required tools are installed and ready to use. We will go over how to run the project on your computer.

Running the Project:

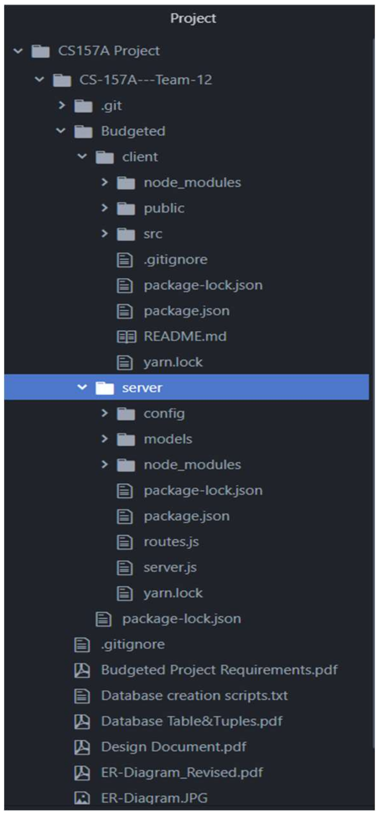
1. Create a project folder on the desktop

2. Open the folder and right-click, then choose "Git Bash Here"

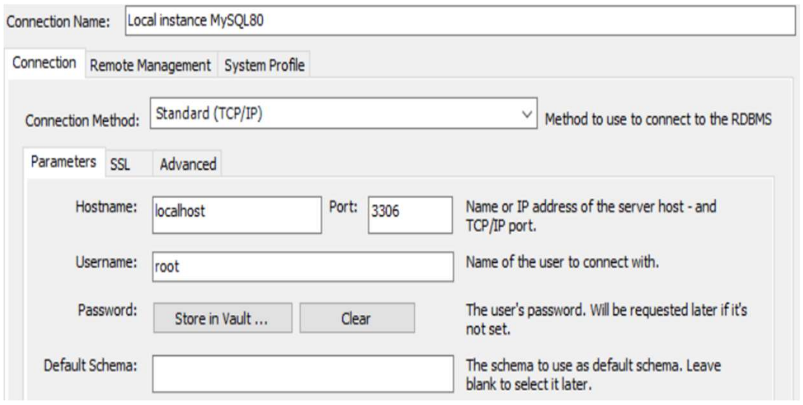
3. Type in "Git Clone https://github.com/mczhou1984/CS-157A---Team-12.git" in prompt. You may need to enter your user name and password.

4. Open Atom > Add Project Folder > Select "CS-157A---Team-12"

5. Open MySQL Workbench 8.0 > Create New Connection > fill in the below information. The password is "password".



```
Project
CS157A Project
  CS-157A---Team-12
    .git
    Budgeted
      client
        node_modules
        public
        src
        .gitignore
        package-lock.json
        package.json
        README.md
        yarn.lock
      server
        config
        models
        node_modules
        package-lock.json
        package.json
        routes.js
        server.js
        yarn.lock
      package-lock.json
    .gitignore
    Budgeted Project Requirements.pdf
    Database creation scripts.txt
    Database Table&Tuples.pdf
    Design Document.pdf
    ER-Diagram_Revised.pdf
    ER-Diagram.JPG
```



Connection Name: Local instance MySQL80

Connection | Remote Management | System Profile

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Parameters | SSL | Advanced

Hostname: localhost Port: 3306 — Name or IP address of the server host - and TCP/IP port.

Username: root — Name of the user to connect with.

Password: Store in Vault ... Clear — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

6. Open the "Database Creation Script" from your project folder and copy all texts.

7. In your MySQL Workbench 8.0 > Local Instance MySQL80 > Create new SQL > Paste the texts you copied and run it. It will create all the required schema for the project.

8. From Start Menu > Open the cmd prompt

9. Navigate to ./Budgeted/Client

10. Type in “npm install”, then “npm start”

11. From Start Menu > Open another cmd prompt

12. Navigate to ./Budgeted/Server

13. Type in “npm install”, then “npm run server”

14. Open your browser at localhost:3000.