CS3354 Software Engineering
Final Project Deliverable 2

DailyBudget

Minh Chung, Jason Nguyen, Chathuri Palanivelu, Chloe Pascual, Jerry Wang

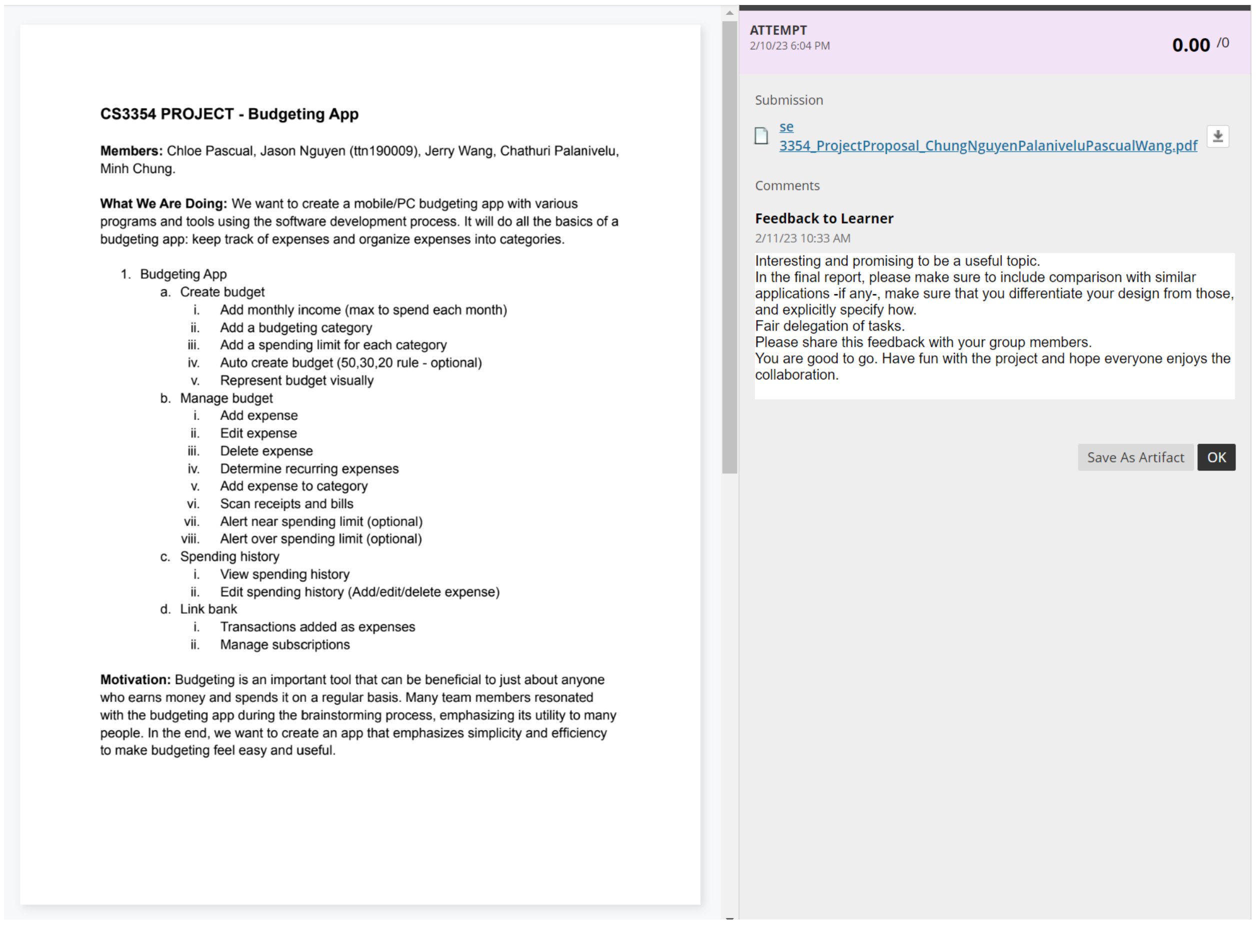
Delegation of Tasks

1. Minh Chung - Class diagram, Hardware costs, Github, Function point
2. Jason Nguyen - Software requirements, Project comparison, Github
3. Chathuri Palanivelu - Use case diagrams, Github, Test case, Function point
4. Chloe Pascual - UI design, Software and personnel costs, Sequence diagrams, Function point, Github
5. Jerry Wang - Software process model, Conclusion, Function point, Github

Overall, everyone contributed well

Project Deliverable 1 Content

Final Project Draft Description with Feedback



CS3354 PROJECT - Budgeting App

Members: Chloe Pascual, Jason Nguyen (ttn190009), Jerry Wang, Chathuri Palanivelu, Minh Chung.

What We Are Doing: We want to create a mobile/PC budgeting app with various programs and tools using the software development process. It will do all the basics of a budgeting app: keep track of expenses and organize expenses into categories.

1. Budgeting App
   a. Create budget
      i. Add monthly income (max to spend each month)
      ii. Add a budgeting category
      iii. Add a spending limit for each category
      iv. Auto create budget (50,30,20 rule - optional)
      v. Represent budget visually
   b. Manage budget
      i. Add expense
      ii. Edit expense
      iii. Delete expense
      iv. Determine recurring expenses
      v. Add expense to category
      vi. Scan receipts and bills
      vii. Alert near spending limit (optional)
      viii. Alert over spending limit (optional)
   c. Spending history
      i. View spending history
      ii. Edit spending history (Add/edit/delete expense)
   d. Link bank
      i. Transactions added as expenses
      ii. Manage subscriptions

Motivation: Budgeting is an important tool that can be beneficial to just about anyone who earns money and spends it on a regular basis. Many team members resonated with the budgeting app during the brainstorming process, emphasizing its utility to many people. In the end, we want to create an app that emphasizes simplicity and efficiency to make budgeting feel easy and useful.

ATTEMPT
2/10/23 6:04 PM
0.00 /0

Submission
se 3354_ProjectProposal_ChungNguyenPalaniveluPascualWang.pdf

Comments

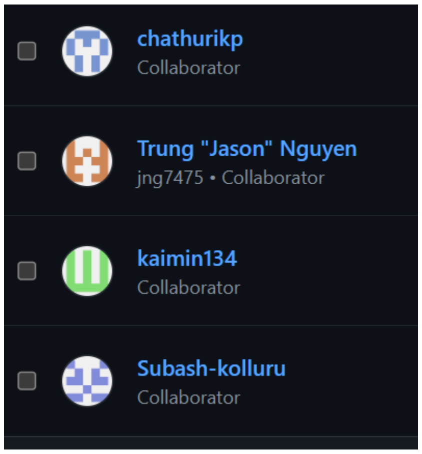
Feedback to Learner
2/11/23 10:33 AM

Interesting and promising to be a useful topic.
In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.
Fair delegation of tasks.
Please share this feedback with your group members.
You are good to go. Have fun with the project and hope everyone enjoys the collaboration.

We will make sure to discuss similarities and differences of our app with other budgeting apps in the final report.

Github

Github Repository: https://github.com/ccmmp/3354-bugetingApp
Added Collaborators:



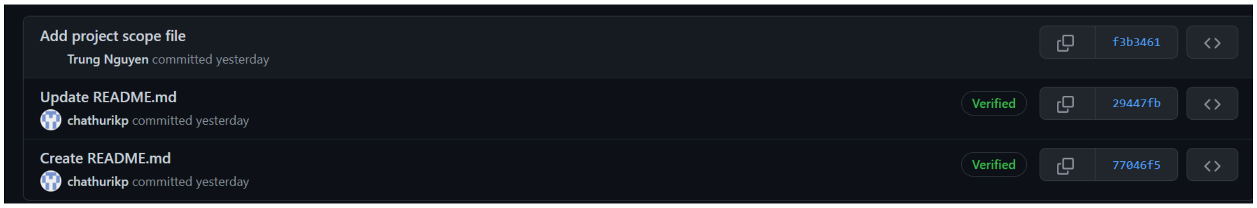
chathurikp
Collaborator

Trung "Jason" Nguyen
jng7475 • Collaborator

kaimin134
Collaborator

Subash-kolluru
Collaborator

Commits to Repository:



Add project scope file
Trung Nguyen committed yesterday — f3b3461

Update README.md
chathurikp committed yesterday — Verified — 29447fb

Create README.md
chathurikp committed yesterday — Verified — 77046f5

Delegation of Tasks*
***For deliverable 1 each member worked on a section and tasks were split evenly**

a. **Chloe**
   i. Design ui/ux, branding, and frontend responsibilities.
   ii. Backend help where applicable.
b. **Jason**
   i. Help with front-end development
   ii. Researching and setting up most suitable database
   iii. Implement APIs to connect app with the database
c. **Chathuri**
   i. Finding and implementing suitable APIs for scanning feature of the app
   ii. Backend
d. **Minh**
   i. Finding how to protect privacy and security when linking bank accounts to the app.
   ii. Frontend
e. **Jerry**
   i. Can help with front-end development
   ii. Develop features for managing expenses, creating budgets, etc.

Software Process Model

We are using an incremental process which will divide the system's functionality into small increments that are delivered one after the other. Each further increment will expand on the previous ones until everything will be updated fully. This will allow us to see what does and doesn't work in the earlier increments, allowing us to fix them before going further. So, we can work in a stepwise manner and fix bugs and issues as they pop up. Since the incremental process will need a clear and complete definition of the whole system, it works for us since we know how our system will operate and its requirements.

Software Requirements

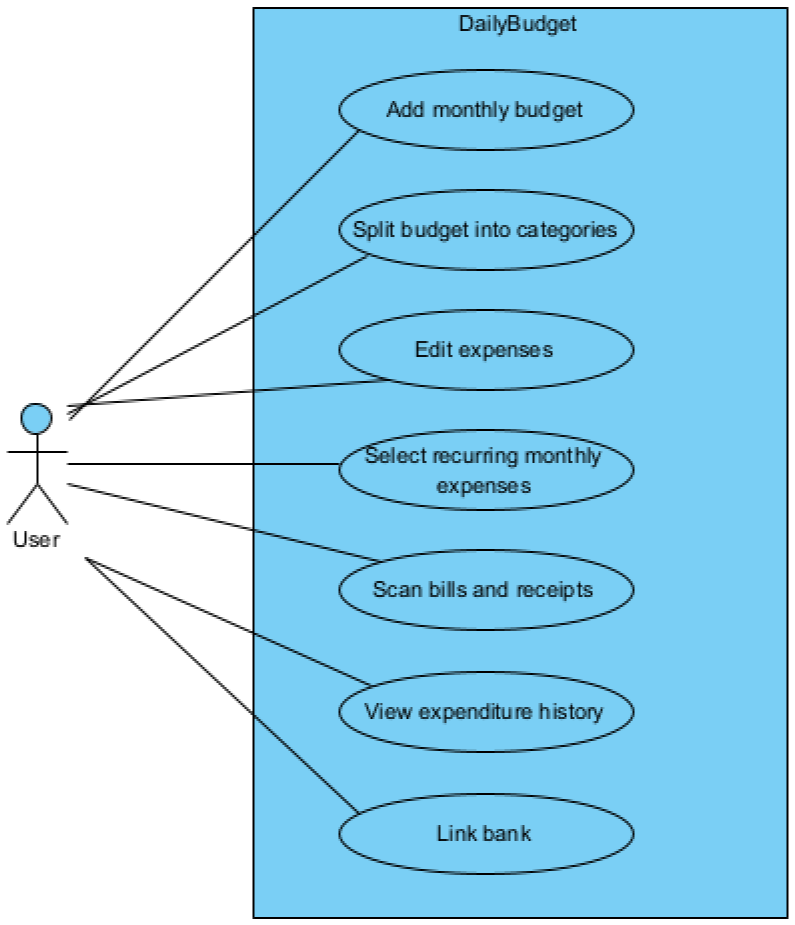
Functional

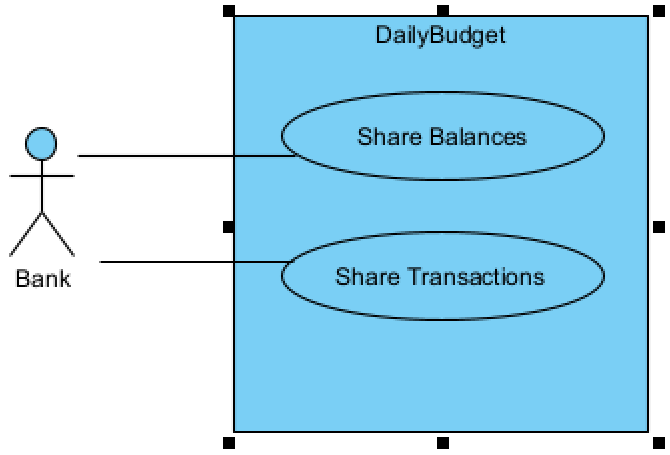
- User registration: The app should allow users to create an account and sign in securely.
- Expense tracking: The app should allow users to track their expenses by category (e.g. groceries, rent, entertainment).
- Budget planning: The app should allow users to set monthly or weekly budgets for each category.
- Notifications: The app should send notifications to users when they approach or exceed their budget limits.
- Goal setting: The app should allow users to set financial goals (e.g. saving for a vacation, paying off debt) and track progress towards these goals.
- Reporting: The app should generate reports and visualizations of spending habits and trends.

Non-Functional

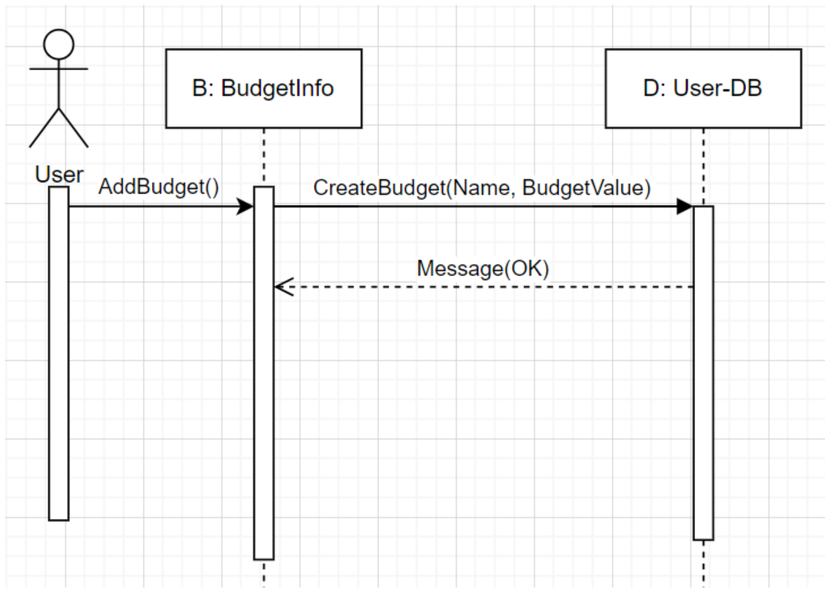
- Security: The app should be designed with strong security features to protect user data.
- Usability: The app should be user-friendly and easy to navigate, with a clear interface and intuitive controls.
- Performance: The app should be fast and responsive, with minimal lag or delay.
- Availability: The app should be available for use on multiple devices and platforms (e.g. iOS, Android, web).
- Reliability: The app should be reliable and stable, with minimal crashes or bugs.
- Scalability: The app should be designed to handle a large number of users and a growing database of financial data.

Use Case Diagrams

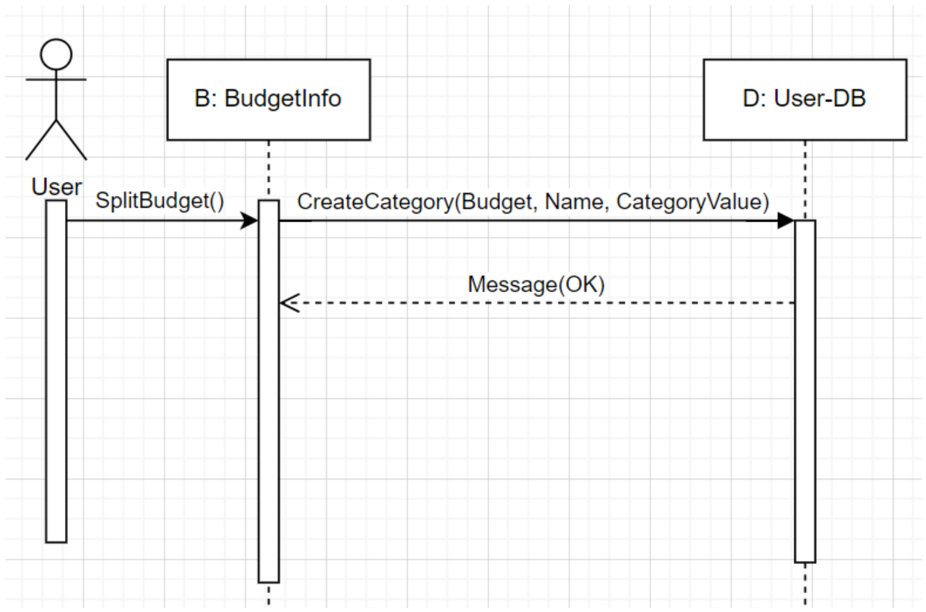




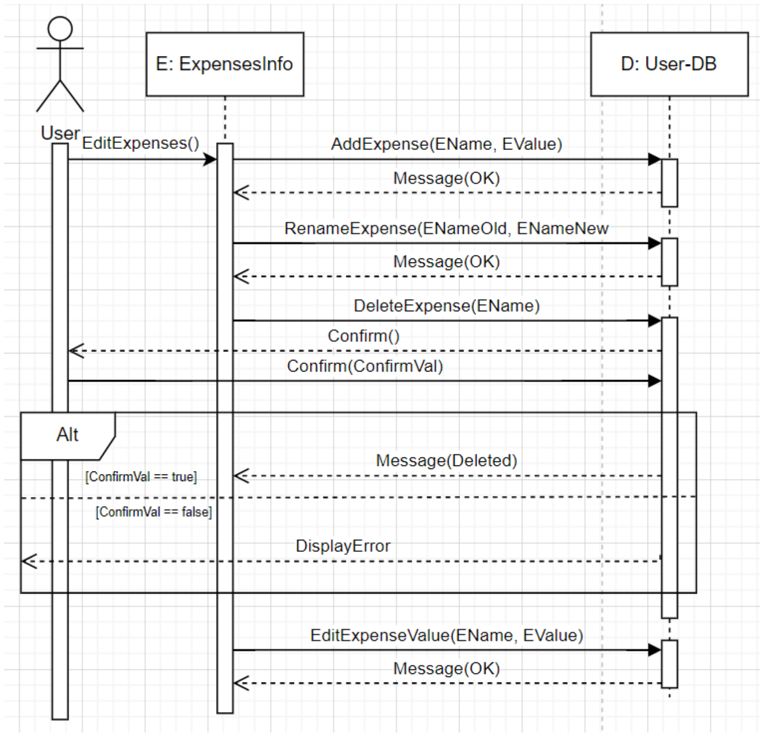
Sequence Diagrams



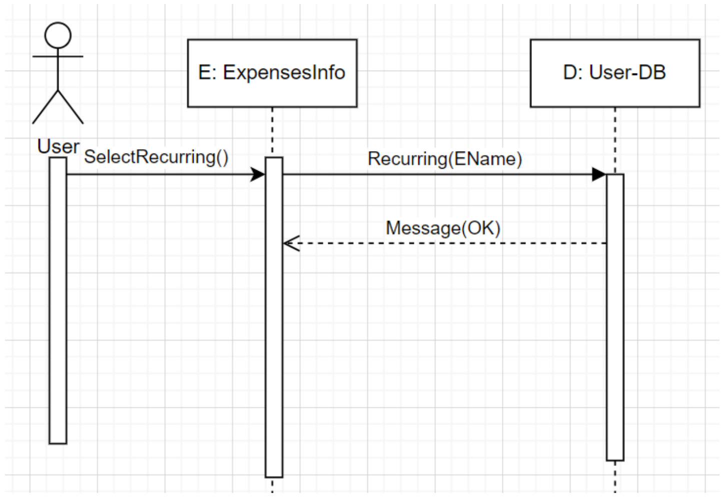
User: Add Monthly Budget



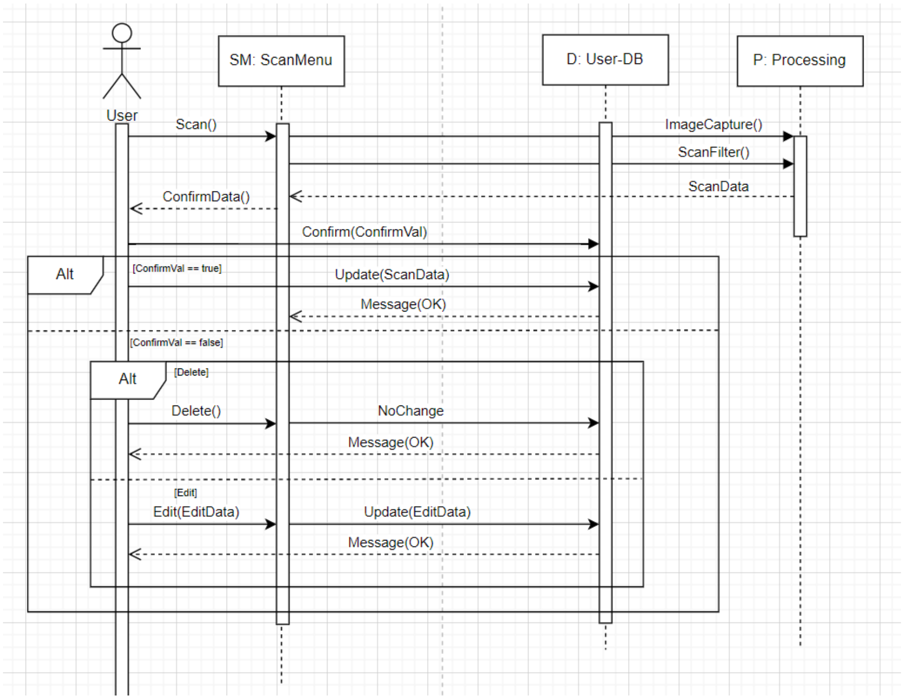
Uder: Split Budget Into Categories



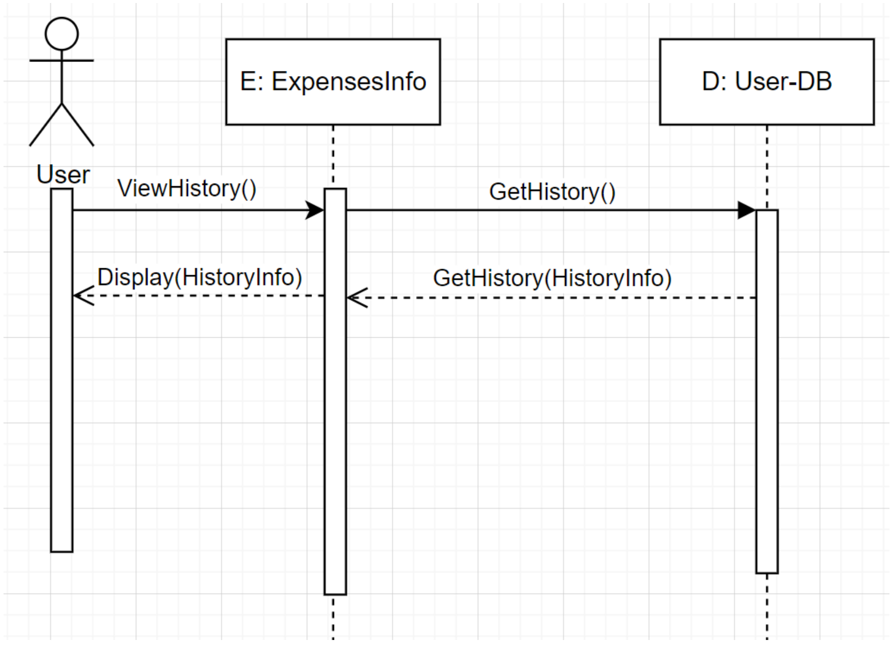
User: Edit Expenses



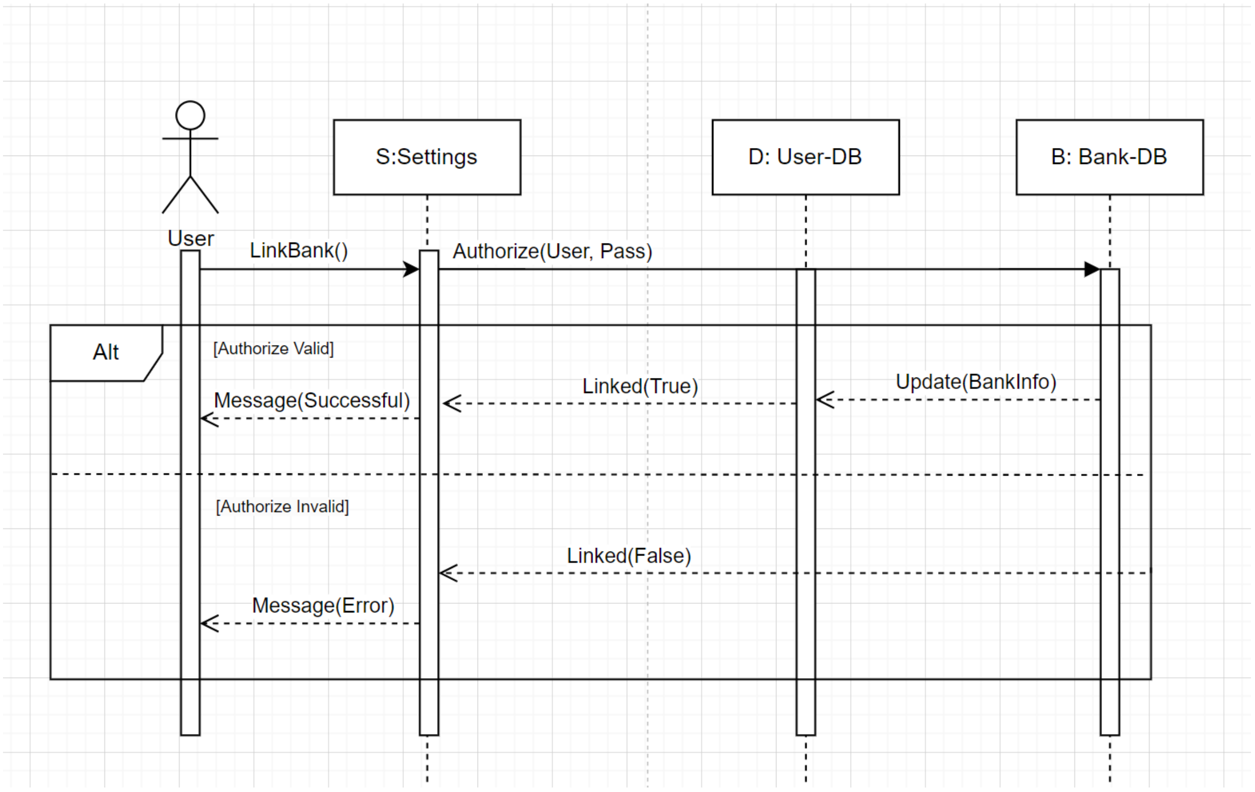
User: Select Recurring Expenses



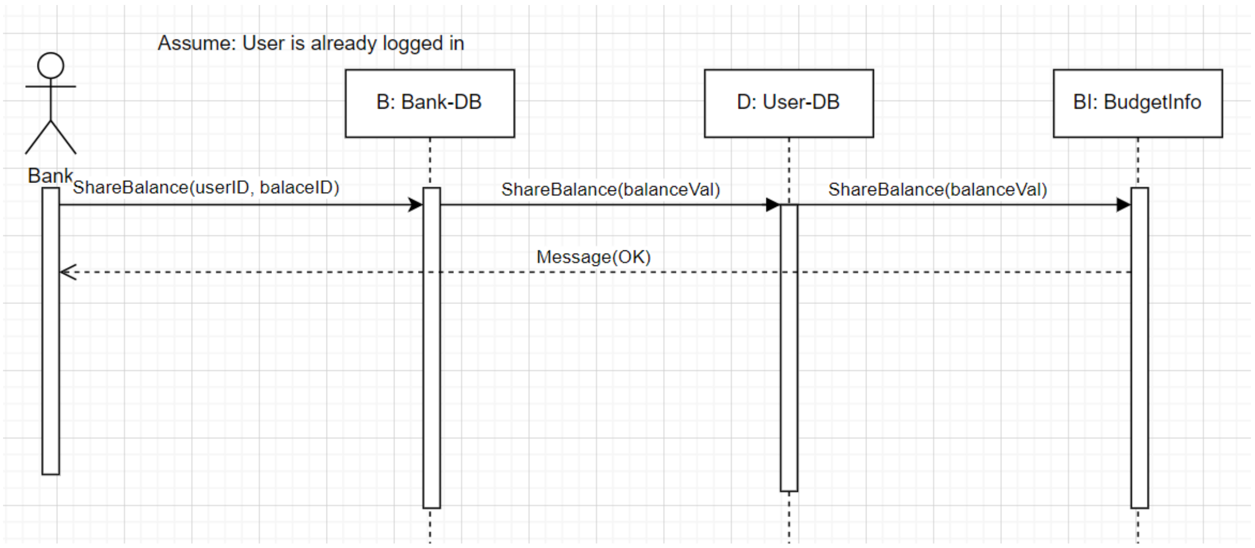
User: Scan Bills and Receipts



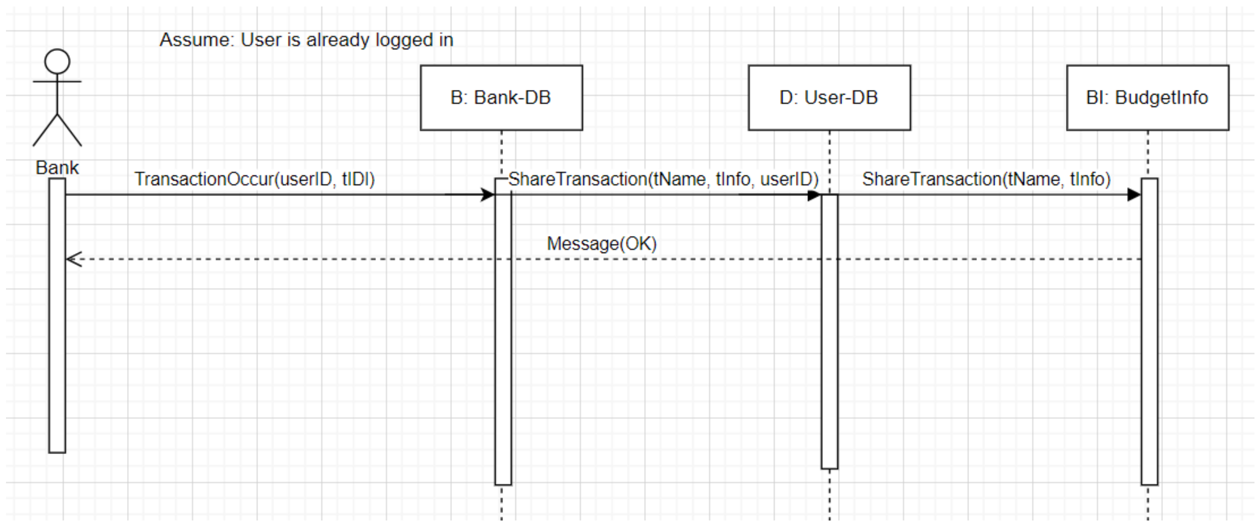
User: View Expenditure History



User: Link Bank

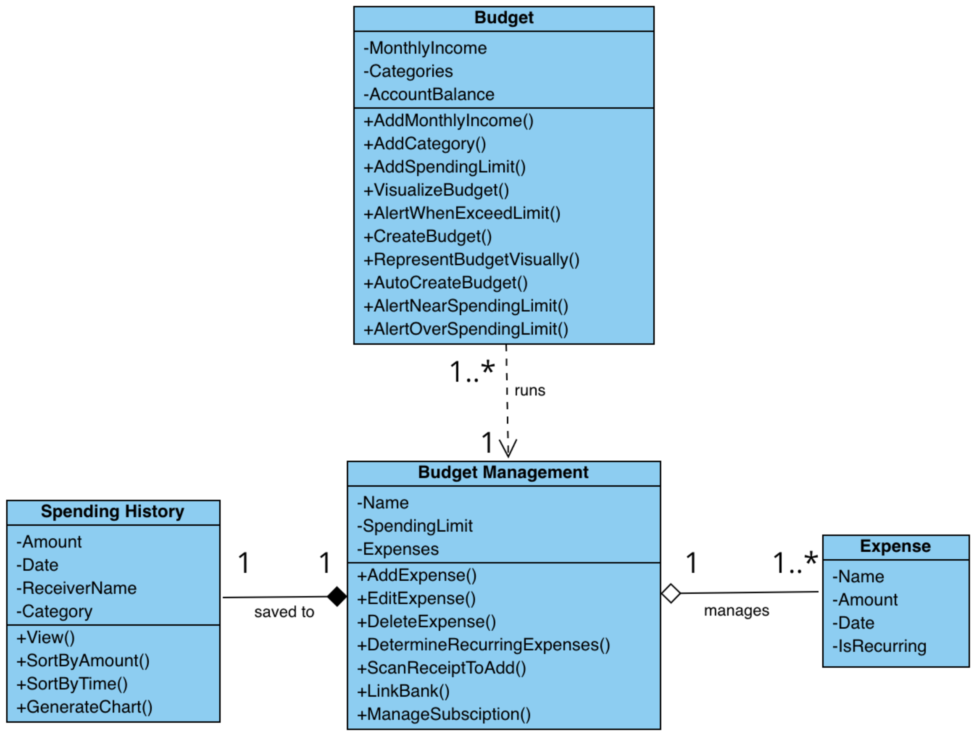


Bank: Share Balances


Assume: User is already logged in
Bank
B: Bank-DB
D: User-DB
BI: BudgetInfo
TransactionOccur(userID, tIDI)
ShareTransaction(tName, tInfo, userID)
ShareTransaction(tName, tInfo)
Message(OK)

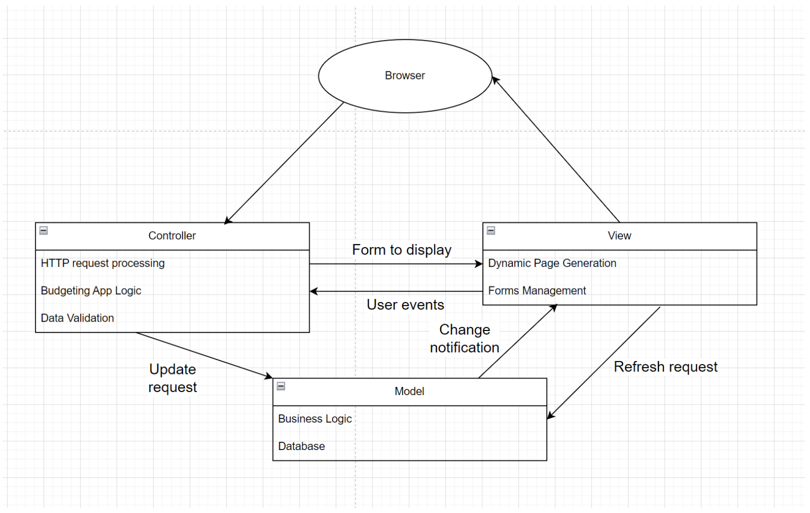
Bank: Share Transaction

Class Diagram



Architectural Design

For the architectural design of this project we decided to go with Model-View-Controller (MVC). This budgeting web-based application handles a good amount of data for each user, displays that data, and that data can be interacted with in a number of ways, because of that MVC is a nice fit.



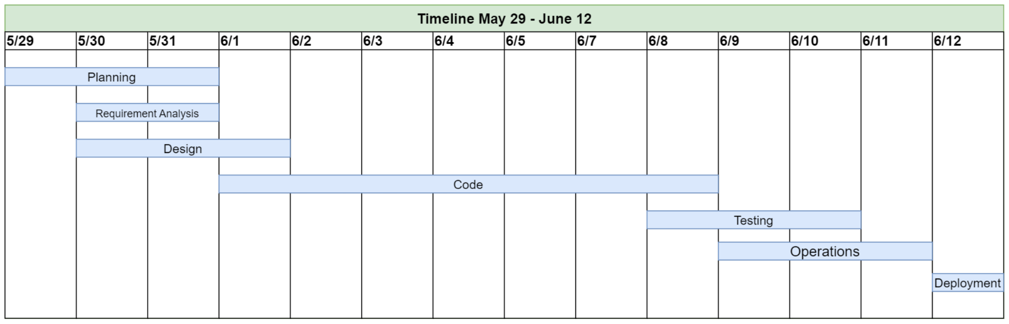
Architecture for web-based budget app

Estimations

1. Project Scheduling

Start Date: **29 May, 2023**

End Date: **12 June, 2023**



Estimated Timeline

Justification: Using the function point estimation model, it was estimated that if the productivity was 10 function points a week (around 2 per end coder), and the gross function point came to be 97. The effort then came out to be around 11 person-weeks, which when divided by a team size of 7 resulted in a **2 week project duration**. In this case a work week is 40 hours/week with weekends off.

2. Function Point Estimation

Function category count

External inputs

username, password, budget name, budget amount, budget category, expense name, expense amount, scan bill, bank information

External outputs

confirmations, updated budget, expenses list, spending chart

External queries

usernames, passwords, budget data, expenses data, account balance

Data fields

user information, bank information, budget categories. actual expenses, planned expenses, account balances, due dates, payment history, net worth, goals, investment, tax-related information, debt repayment, savings goals, currency exchange rates, reminders

External inferences

server, hardware, UI, OS, devices

Determine complexity and gross function point

	Function Category	Count	Complexity			Count x Complexity
			Simple	Average	Complex	
1	Number of external inputs	9	7	1	1	9x7=63
2	Number of external outputs	4	2	2	0	4x2=8
3	Number of external queries	5	3	2	0	5x3=15
4	Number of data fields and relational tables	17	4	8	5	4x2=8
5	Number of external interfaces	4	1	1	3	3x1=3

GFP = 63+8+15+8+3 = **97**

Processing complexity

Does the system require reliable backup and recovery? 5

Are data communications required? 4

Are there distributed processing functions? 2

Is performance critical? 3

Will the system run in an existing, heavily utilized operational environment? 4
Does the system require online entry? 4
Does the online data entry require the input transaction to be built over multiple screens or operations? 4
Are the master files updated online? 3
Are the inputs, outputs, files, or the inquiries complex? 4
Is the internal processing complex? 3
Is the code designed to be reusable? 3
Are conversion and installation included in the design? 1
Is the system designed for multiple installations in different organizations? 3
Is the application designed to facilitate the user? 5

TOTAL: PC = **45**

Processing complexity adjustment

PCA = .65 + .01(45) = **1.10**

Function point

FP = GFPxPCA = 97 * 1.10 = **106.7**

Estimated Effort

E = FP/Productivity = 106.7/10 = 10.67 ≈ **11 person - weeks**

Project Duration

D = E/Team Size = 11/7 ≈ **2 weeks**

3. Hardware Cost

Our team members already have personal computers that meet the minimum requirements for app development (e.g. sufficient processing power, memory, storage, and graphics capabilities). In terms of mobile app version testing, we have some team members who use Android phones, and some team members who use iOS phones, so we can easily test our product on two major operating systems of mobile phones. Thus, the only hardware product we need is a server.

ThinkSystem SR550 Rack Server: **$2,553.20**.

TOTAL: **$2,553.20**.

4. Software Costs

GitHub Team = $4/month/user x 7 team members = $28
Slack = $7.25/month/user x 7 team members = $50.75
Azure (*Dev Ops, Storage, App Service, Monitor*)/month = $161.42
Linx *Business 1 = $99/month* = $99
Total Software Cost = 28+50.75+161.42+99 = **$339.17**

5. Personnel Costs

5 end coders x $45/hour x 40 hours/week x 2 weeks = $18,000
1 scrum master x $40/hour x 40 hours/week x 2 weeks = $3,200
1 business analyst x $36/hour x 40 hour/week x 2 weeks = $2,880
Personnel Training x 7 employees x $35 x 72 hours of training = $17,640
Total Personnel Cost = 18000+3200+2880+17640= **$41,720**

6. **Final Cost, Effort, and Pricing Estimation**

i. The cost is calculated taking into account the previously calculated hardware, software, personnel, and travel costs:

Hardware = $2,553.20
Software = $339.17
Travel = $0
+ Personnel = $41,720

Total Cost = $44,612,17

ii. Effort

Effort: 2 40 hour work weeks with 7 personnel

iii. Pricing

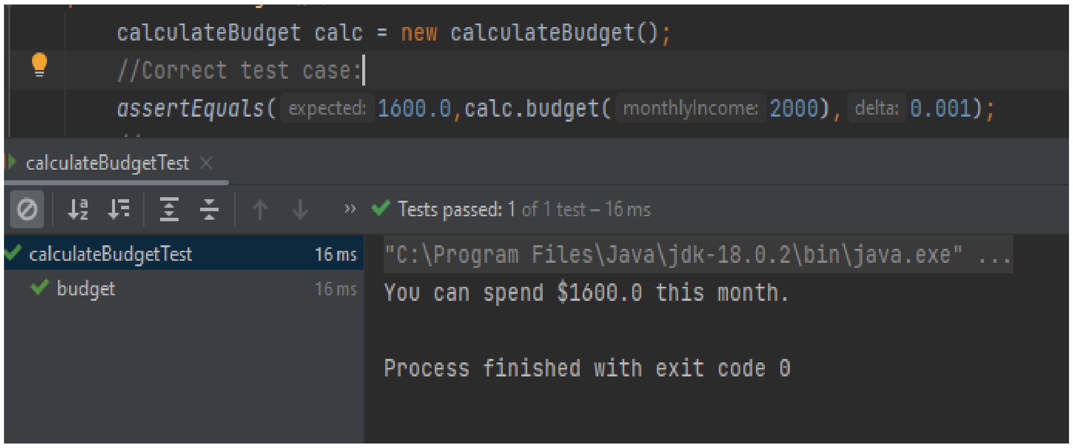
Free Plan = $0
Premium = $5 monthly or $54 annually
Advertising = $0.5 - $12 per advertisement

7. Testing

Testing plan for calculating the budget for the user:
In this test plan, we will allocate the money the user earns every month into three categories - savings, spending needs, and spending wants. So every month, the user can save and spend a certain amount of money based on the calculations the app does.
In the uniting testing, we will be testing the budget under the assumption that the monthly salary that is passed to function is greater than zero. We have two main test cases:

1) When the calculated budget is the same as the expected budget cost (correct calculation)

This results in the test case passing, and we can proceed to use the function for other calculations.



```
calculateBudget calc = new calculateBudget();
//Correct test case:
assertEquals( expected: 1600.0,calc.budget( monthlyIncome: 2000), delta: 0.001);
```

calculateBudgetTest

Tests passed: 1 of 1 test – 16 ms

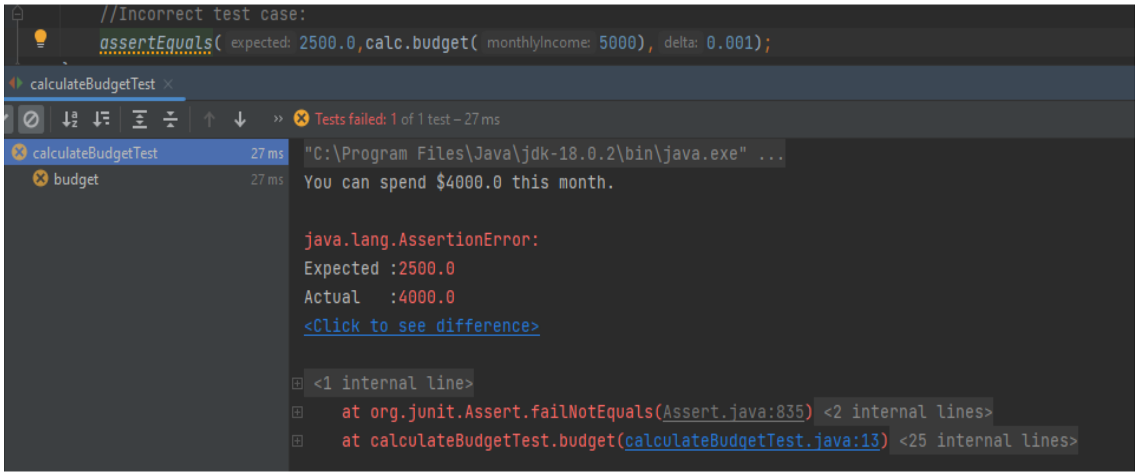
✓ calculateBudgetTest 16 ms
✓ budget 16 ms

```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" ...
You can spend $1600.0 this month.

Process finished with exit code 0
```

2) When the calculated budget is not the same as the expected budget cost (incorrect calculation)

This results in the test case failing, and we need to recheck the expected budget cost possibly.



```
//Incorrect test case:
assertEquals( expected: 2500.0,calc.budget( monthlyIncome: 5000), delta: 0.001);
```

calculateBudgetTest

Tests failed: 1 of 1 test – 27 ms

calculateBudgetTest 27 ms
budget 27 ms

```
"C:\Program Files\Java\jdk-18.0.2\bin\java.exe" ...
You can spend $4000.0 this month.

java.lang.AssertionError:
Expected :2500.0
Actual   :4000.0
<Click to see difference>

<1 internal line>
    at org.junit.Assert.failNotEquals(Assert.java:835) <2 internal lines>
    at calculateBudgetTest.budget(calculateBudgetTest.java:13) <25 internal lines>
```

The test code can be found in the GitHub repository.

8. Project Comparison

Our application is going to be focusing on a user-friendly interface, fewer but higher standard features, and affordability. Compared to one of the current popular budget apps such as "Mint", we focus on providing a smooth and easy interaction process for users. This will encourage more people who are not used to keeping track of their expenses to be more financially responsible. As our application focuses on fewer features, we will be able to ensure the highest quality for each one and provide a more stable connection compared to other applications. Moreover, our application will have a free version and a premium version with a lower starting price. We understand the need to save money using a budget application, so our application will make sure the users have that benefit.

9. Conclusion

By working together, we were able to establish a plan to build DailyBudget in two weeks' time, with proper resources allocated to ensure that the process goes smoothly. We also had to brainstorm ways to make the application helpful to people who are new to budgeting and maintaining their finances. Thus, we considered user interface and accessibility to be important for our application. We also made sure to consider whether or not we'd need to make the application include in-app purchases like a premium subscription that allows for unlimited categories rather than a select number for the free version, or extra features that would enhance the user's experience. This would be mostly to ensure that the project can generate revenue to fund our resources in order to build and maintain the project. We also made sure to consider different mobile phone OSs, namely Apple and Android, when it came to testing our product. We concluded that our project would have several details that would make the application much more user-friendly while also allowing our team to work in a timely manner to finish within our estimated planned time.

10. References

N. Etzel, "Mint Review: Pros, cons, and more," *The Motley Fool*, 22-Feb-2023. [Online]. Available: https://www.fool.com/the-ascent/personal-finance/mint-review/. [Accessed: 21-Apr-2023].

S. Vaughan-Nichols, "Microsoft Azure Review," *PCMAG*, 12-Jul-2017. [Online]. Available: https://www.pcmag.com/reviews/microsoft-azure. [Accessed: 21-Apr-2023].