**Project Title:** Financial Bloom: Budgeting App with Habit Growth Integration

**Project Description:** Financial Bloom is an innovative finance-tracking app designed to help users manage their income, expenses, debt, savings, and personalized budgets. With a unique gamified approach, users' positive financial decisions contribute to the growth of a virtual garden, turning responsible spending and saving habits into a visually rewarding experience. This app simplifies financial planning, motivates users to maintain good financial habits, and encourages long-term financial health.

**Project Overview Statement:** Financial Bloom is a budgeting app that helps users manage their finances while fostering positive habits through gamification.

**Problem/Opportunity:** The average American carried a total debt of $104,215 in Q3 of 2023, according to *Business Insider*. This debt and poor financial planning make it difficult for many Americans to build a proper savings base. For Americans under 35, the average savings balance should be $20,540, but only 41% of Americans could cover a $1,000 emergency purchase. By gamifying responsible financial behaviors, Financial Bloom aims to motivate users to develop sustainable financial habits, empowering them to reduce debt, increase savings, and be more mindful of their spending choices.

**Goal:** To create a user-friendly application that encourages financial growth and responsibility using the metaphor of a garden. Responsible spending and saving habits will produce a rewarding game-like experience.

**Objectives:**

1. Users can track their income, expenses, debt, and savings. Implement a tracking system for income, expenses, debt, and savings.
2. Users can create and implement a budget plan. Design a budget plan capable of user customization.
3. Positive financial decisions contribute to the growth of a virtual garden. Link the proposed tracking system to an aesthetically pleasing dynamic virtual garden.
4. Impulse purchases are tracked and users are notified if they don’t adhere to their personal goals. Establish a notification system for increased active-user retention.

**Success Criteria:**

1. Achieve a 20% increase in user savings within one year of regular app usage.
2. Facilitate a 20% reduction in total user debt within the same period.
3. Support users in successfully paying off debt or reaching a significant financial milestone. A large percentage (60-80%) of users will have been within budget 70% of the year.
4. Ensure that 80% of users report improved financial habits after one year of consistent app engagement.

**Assumptions, Risks, Obstacles:**

1. Assumption: The gamification aspect will increase user engagement and retention by offering a visually rewarding experience.
2. Risk: Storing user financial data requires robust security measures, including encryption and secure authentication protocols. No vulnerable ports will be left open to prevent unauthorized access.
3. Obstacle: Users may struggle to follow recommendations or make changes based on app notifications, reducing the system's effectiveness.
4. Assumption: Users will be sensitive and supportive of the financial plan given by the application. (Could be a bit redundant with #3, up to y’all if we keep it)
5. Risk: Users may lose motivation if their virtual garden withers or dies, negatively impacting long-term engagement. Solutions such as recovery mechanisms or progress resets can mitigate this risk.

**Roles:**

* UI/UX - Daniela and Laura
* Security - Mikaela and Laura
* HTML - Mikaela and Chris and Laura
* Backend - Daniela and Temo

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**To integrate banking information and advanced budgeting features in your app:**

1. Use a Banking API (Open Banking or Plaid)

To access and display users' banking information securely, you'll need to integrate a third-party financial API. Common options include:

* Plaid: Provides access to user bank accounts, transaction data, and balance information.
  + <https://plaid.com/docs/sandbox/>
  + Reportedly Easiest to integrate
  + Best for financial apps/startups
* Yodlee: Another leading service for financial data aggregation.-
  + <https://www.yodlee.com/fintech/developers/developer-portal>
  + Best for loans
* Finicity (Mastercard): Trusted for financial services and compliance
  + .<https://www.finicity.com/manage/>
  + Best for international customer base

Steps for Integration:

* Sign up with a service like Plaid and follow their API documentation for secure integration.
* Implement OAuth2 for secure user authentication when they connect their bank account.
* Ensure compliance with PCI-DSS (Payment Card Industry Data Security Standard) for handling financial data.

2. User Budget and Spending Tracking Integration

Once the bank data is accessible, you can build features to automatically:

* Pull checking and savings balances.
* Import and categorize transactions.
* Log transactions into budget categories defined by the user.

Steps for Budget Implementation:

* Develop logic for users to set budget categories (e.g., groceries, entertainment).
* Compare actual spending (based on transactions) with budget targets.
* Provide visual insights (charts, graphs) on spending trends.

3. Recurring Payments Feature

We can enable customers to add recurring payments, such as loan payments, subscriptions, or regular deposits, into their budget.

Implementation:

* Allow users to define custom recurring expenses or income (e.g., every month or week).
* Automatically include these in budget calculations.
* Notify users of upcoming recurring payments or unexpected deviations from their plan.

4. Security Considerations

* Data Encryption: Encrypt all sensitive data in transit and at rest.
* Secure Authentication: Use OAuth2 and multi-factor authentication (MFA).
* Compliance: Ensure the app complies with GDPR and CCPA for user privacy.
* RELEVANT NIST PUBLICATIONS:
  + <https://csrc.nist.gov/pubs/sp/800/171/r2/upd1/final> - Data Security Framework
  + <https://pages.nist.gov/800-63-3/> - Secure Auth Guidelines
  + <https://csrc.nist.gov/pubs/sp/800/207/final> - Zero Trust Architecture
  + <https://csrc.nist.gov/pubs/sp/800/88/r1/final> - Secure Deletion for Financial Info

5. User Experience Enhancements

* Offer a feature to flag unusual transactions.
* Provide motivational features when users meet budget goals (e.g., growing the virtual garden).
* Enable goal-setting for future savings or debt repayment milestones.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Random project ideas/notes (BRAINSTORMING):**

Users: 16+, user-friendly interface for all age ranges. Users will more than likely be people who are looking to improve their finances.

Language: Java, JS, SQL, HTML, CSS

Will the user input all information or is there a way/does anyone know how to import banking information (this would be a security risk)- check for vulnerabilities if using a web-based platform. OR we could ask the user if they want to import banking information or continue entering information themselves. (Users can link their bank for real-time updates to their budget)

Platform: web-based, app, or both?

Questions:

1. What happens if the garden dies?
2. Are we going to have the user directly interact with the garden or have their actions be more of the interaction and the user only be able to check the status of their plants and be able to maneuver around their garden?
3. Should we have preset budget categories or allow the user full customization?

UI/UX:

* Garden Color Scheme: greens, blues, pinks, yellows(?)
* Budget Color Scheme: green and white with small pops of blues, pinks, and yellows
* Sleek design with pixelated

Non-functional requirements:

* The system logs out the user after a certain amount of time of inactivity
* The system has a hash table that will store log-in information.
* The system sends push notifications with tips/ideas/encouragement
* The system has different plants for different categories of expenses
* Plants get smaller as the user’s priorities move away/stray.
* The system has a tutorial feature that users can look at before signing up
* If user closes their account, system deletes log-in info automatically
* If someone enters an existing username, “username already exists” error will pop up.
* Password recovery.

Home page (Not Features – some can be turned into non-functional features):

* Website name & logo - located at the top
* Description
* Log-in page: sign up/log-in options will be combined with the *Welcome* page that talks a little bit about the website
* Tutorial - Wouldn’t this be a feature? Users can go through a tutorial to understand how to use the application.

Features: Critical, Important, Useful

* Users log into the system with MFA (C)
* Users may log out at any time through a log-out button (C)
* User can update, add, and delete their own information (whether that be bank accounts, budget templates, etc) (C)
* Users can edit/customize the budget template (U)\*+
* Users can customize their dashboard (U)
* Users can choose between a few different savings plans. (I)
* Users can close their account. (U)

Dashboard page (after the home page- the very next page after the user logs in) (functional requirements/features):

* Users create a monthly budget which they can modify at any time to better reflect their goals. (U)
* Users follow some preset budgets or make their own. (C)
* Users set up milestones that they contribute to within a budget. These milestones will either be debts or personal things to save up for. (C)
* Users choose which type of plant represents their milestone. (I)
* Users apply other forms of income to a monthly budget (side gigs, odd jobs, gifts, etc). (I) (Temo Galindo personally believes this to be a U)
* Users deduct emergency money from savings/the budget. (I)
* Users can compare or share their gardens. (U)
* User celebrate moving towards their goals by “watering” their garden and logging their progress (U)
* Users log any debts they wish to keep track of, which will show up in the garden as a rock (U)
* Users can opt into daily tips to better keep to their budget (U)

UML Rough Draft

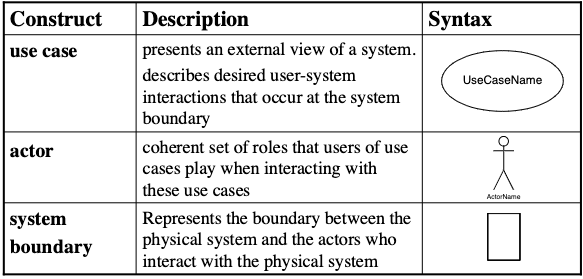
Unfinished- Chris <https://lucid.app/lucidchart/ff0aeeaf-618a-43f6-ab96-6662c9c07f5e/edit?viewport_loc=-706%2C607%2C2478%2C2947%2C0_0&invitationId=inv_1181358d-d448-4368-b4ad-58b9f6490392>

Notes from 2/11 (from Temo)

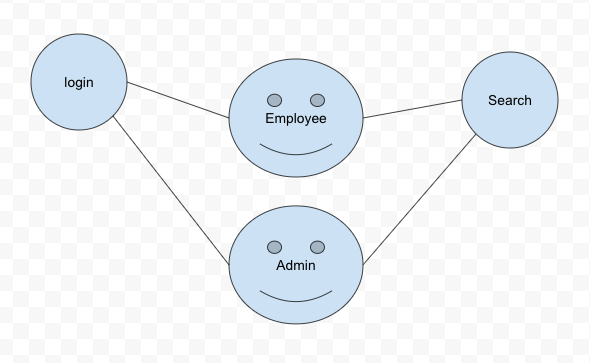
* First went over the quiz, specifically the answers pdf under week 5 module.
* We will have a quiz next week on requirements - actors, elicitation and feature priority, as well as what we go over today.

Actual notes

* We can have requirements, but the implementation can be completely very different.
* For the phone book system, some implementations may use a 2 factor authentication, others might use biometrics, some might use a website and server based security.
* This is where meeting with the user is important - detailed requirements come from the user. If there are no detailed requirements for it we have more freedom in how we can accomplish it.
* UML (Unified Modeling Language) is a graphical way to model our requirements - we will be using Use Case Models, which are a variation for AGILE.
* UCM (Use Case Model) is how we model requirements, as it emphasizes the behaviour as seen by the actor (outside user).
* An interaction MUST be: An actor requests something from the system AND the system responds in some way. If the actor requests but there is NO response, then it is not an interaction.

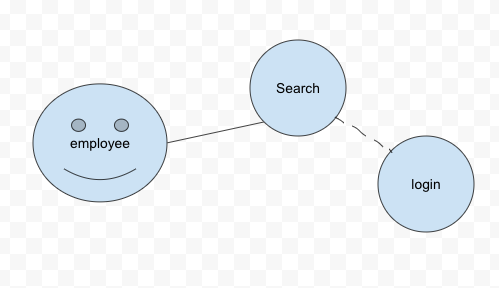


* Every use case must be UNIQUE and describes an interaction. EX. “login” as a use case in Phone Book system- between an employee and an admin, both must login, so we would only have it ONCE as a use case.
* Actors should also be UNIQUE, even among multiple use cases. EX. “Employee” and “Admin” are both actors for Phone Book - both would login as well as both using the “search” function, so we need to only have one copy of each. (pic. 1)



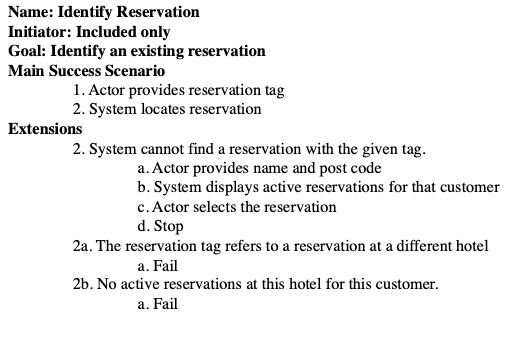
Picture 1

* We use solid lines for actors and use cases, but dotted lines for includes between use cases. EX. In order to search we must be logged in. (pic 2)



Picture 2

* First step for UCM is to identify the actors - who is going to use the system? What are their available use cases? Is there overlap?
* Second step is to identify use cases and a brief description of their goals. For each use case we make a name, who is the initiator and a short goal. EX. Use case is “Make a reservation,” initiator would be a reservation maker (they get the room for themselves or on someone else’s behalf), and the goal is to reserve an available room.
* Third step is to identify the actor and use case relationship - basically link steps 1 and 2.
* Fourth step is to outline individual use cases - “Single numbered sequence of steps that describe main success scenario” (success scenarios) and “Extensions if any” (what happens if things go wrong). EX. for the login use case, logging in successfully (success scenario), and if the user fails or forgets their password (extensions).
* Fifth step is to refine the use case to include alternate flows, including exceptions.



Have our success scenarios as well as extensions where the system would run into errors (extension 2 is if success scenario 2 runs into an error, 2a is if a runs into an error, etc.).

* We then went over an exercise where we made actors, use cases and drew the relationships for the phone company description.

02/13/2025 Notes

* There are two refinements to the use cases: extensions or inclusion. Inclusion means that one use case is being used by another used case. For example, in phonebook: users have to log in before they can do any other feature, so the login is an included use case. We cannot use association to link used cases together (use dotted lines instead).
* Extensions are exceptions that need to be handled i.e. failed scenarios.
* In the Agile method, we start with the product backlog.
* We document the features as user stories.
* We always start with the highest-priority requirements when beginning with the sprint backlog. Requirements are specified as user stories. We will convert user stories into requirements.
* User story: requirement/request written from the perspective of the end user. You can easily convert a feature into a user story.
  + “As a <role> I want to achieve <goal> (so that <reason>)” [providing a reason is optional]
  + Example: As a client of the hotel I want to cancel a reservation.
  + Class participation: Write a user story for the feature “Administrator adds an entry.” and then an acceptance criteria. Model a success scenario.
  + User story: As an admin, I want to add an entry so that every employee is included in the phone book.
  + Acceptance criteria with success scenario: “Given”/”When”/”Then”:
    - Given: The admin is on the portal to add an employee phone number
    - When: the admin enters a new phone number
    - Then: the system updates the employee's phone number
* We use acceptance criteria to provide details to the user story.
* Story refinement: “Given-When-Then” structure. The given condition must be true in order to continue. “When” the actor does an action. “Then” is what the system does as a result.

Agile encourages change and can handle changes, however, we have constraints in how we handle these changes. When allocating an item to a sprint, you cannot change the allocated item within the sprint. It can be changed before or after, but NOT DURING a spring. (\*\*This will be a question on either the quiz or the test\*\*)

**Activity 4: Use case modeling and user stories**

*Sprint 0:*

* ~~Set up a GitHub~~
* Prototype two high-priority user stories
* Wire-frame
* Select development technologies
* Create an ERD diagram
* Create a relational Schema
* Sketch architectural design
* Sketch module design for the user stories