Name: Alexander Ray

Title: *Cinnamon: Money Management in a Spicier Package*

References: Jacob Hallberg for idea on encapsulating user information and addresses.

Both course Teaching Assistants for help structuring my class diagram, understanding requirements.

Wireframe Mockflow for UI Mockups

Visual Paradigm for class diagram

https://en.wikipedia.org/wiki/Strategy_pattern

https://en.wikipedia.org/wiki/Factory_method_pattern

Project Summary: *Cinnamon* is a personal money management app–akin to Mint–to help track day-to-day spending from different user-defined accounts, including support for external report generation. This project will facilitate smarter, more conscientious spending habits from it's users and will incubate a goal-oriented mindset to money management through its explicit, no-nonsense interface.

Specifically, *Cinnamon* will provide utilities for purchase tracking in both the retail and dining spaces. Factoring in information about the user including location and income, *Cinnamon* can track spending, provide up-to-date information, and generate CSV and JSON reports for external consumption–possibly with different inclusion options.

Project Requirements:

Use Case ID	Use Case Name	User Requirement
UC-1	Sign Up	As a user, I want to be able to sign up for *Cinnamon* and set attributes about myself including location and income so that my information can be included in generated reports
UC-2	Log In	As a user, I want to be able to log in to *Cinnamon* so that I can access my spending information
UC-3	Log Out	As a user, I want to be able to log out of *Cinnamon* so that my spending information is no longer visible
UC-4	Add Account	As a user, I want to be able to add a new account so that I can track spending out of individual sources
UC-5	View Spending	As a user, I want to be able to view each spending instance so that I can track my everyday expenses
UC-6	Log Spending	As a user, I want to be able to log a spending instance–including the type of spending, amount, account, and date–so that they're available as a list to view online or in external reports
UC-7	Generate Report	As a user, I want to be able to generate reports of spending so that I can access this data with external programs and tools

UI Mockups and Class Diagram

Notes:

After extensive conversations with both TA's, I settled on separating most controller logic out into controller classes instead of keeping functionality within a model class. An example of this is having a `AccountController` to handle the addition of a new account instead of leaving that functionality inside `User`. As this method is passed a `User` object, it's intended to do something along the lines of `userObj.getAccounts.add()` within that method. This allows for a separation of concerns while encapsulating data and also fitting REST APIs in an object-oriented project.

Also–related to controllers–after speaking with both TAs we decided controllers do not need state in this case and therefore do not have attributes.

In instances where I do not believe an attribute will be able to be altered after the instantiation of an object, I did not include a setter.

I chose to use aggregation vs composition by considering whether or not the object that is "in" another object could persist after the deletion of the owner. Also, I used dependency lines a lot to indicate uses of other classes as method parameters or return types.

In general, I was seeking to lay out the framework for a Factory Method design pattern for spending instances and the Strategy design pattern for generating reports (as we did in class). In the future, I'm considering Memento for saving state and/or possibly Decorator to add functionality to report generation.

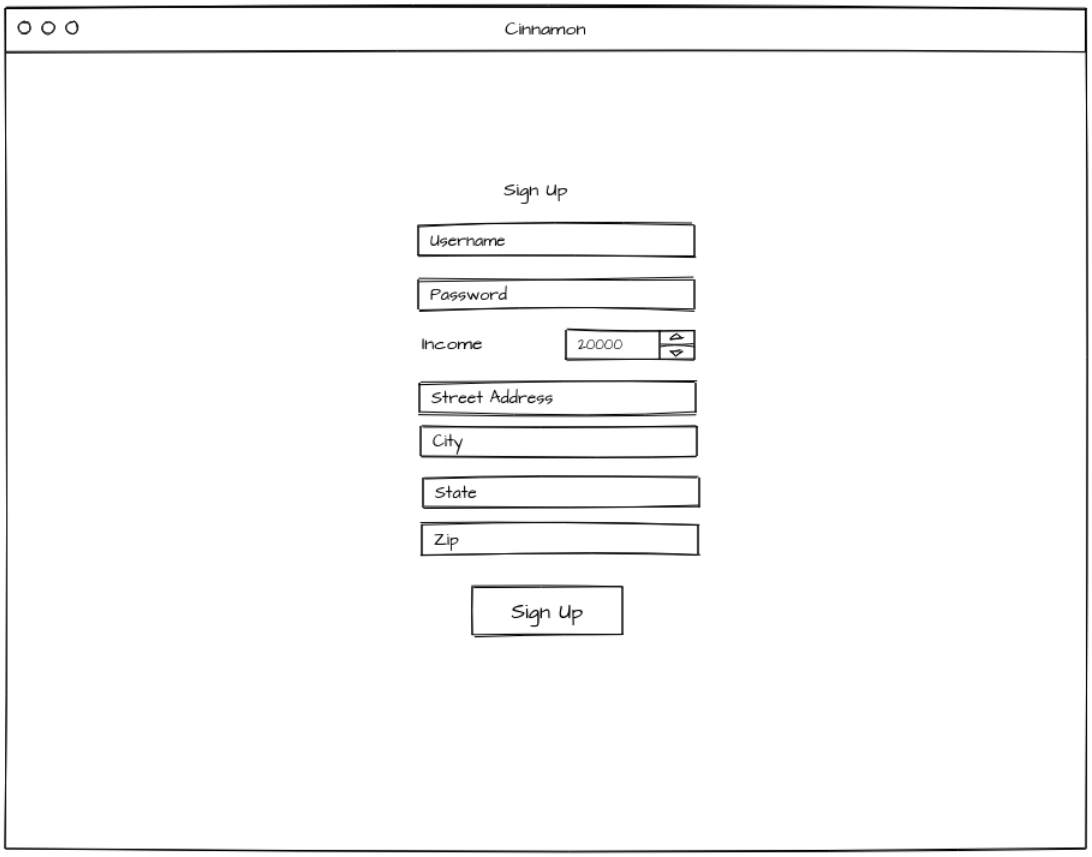


Figure 1: UI Mockup for signup page. User adds username, password, and all necessary information. After this page, there are no options in current mockups to alter this data, though there's the possibility of adding a settings page.

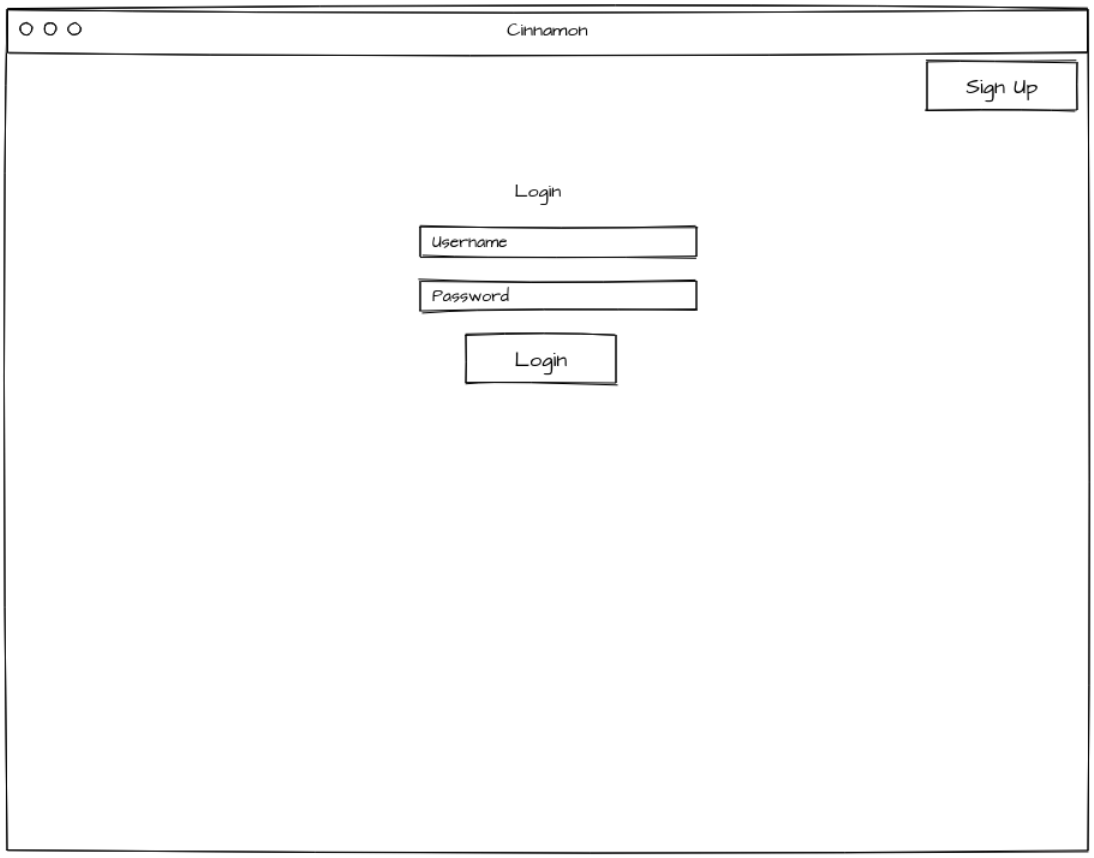


Figure 2: UI Mockup for login page. Mostly felf-explanatory–this page is accessed after a signup, logout, or by default.

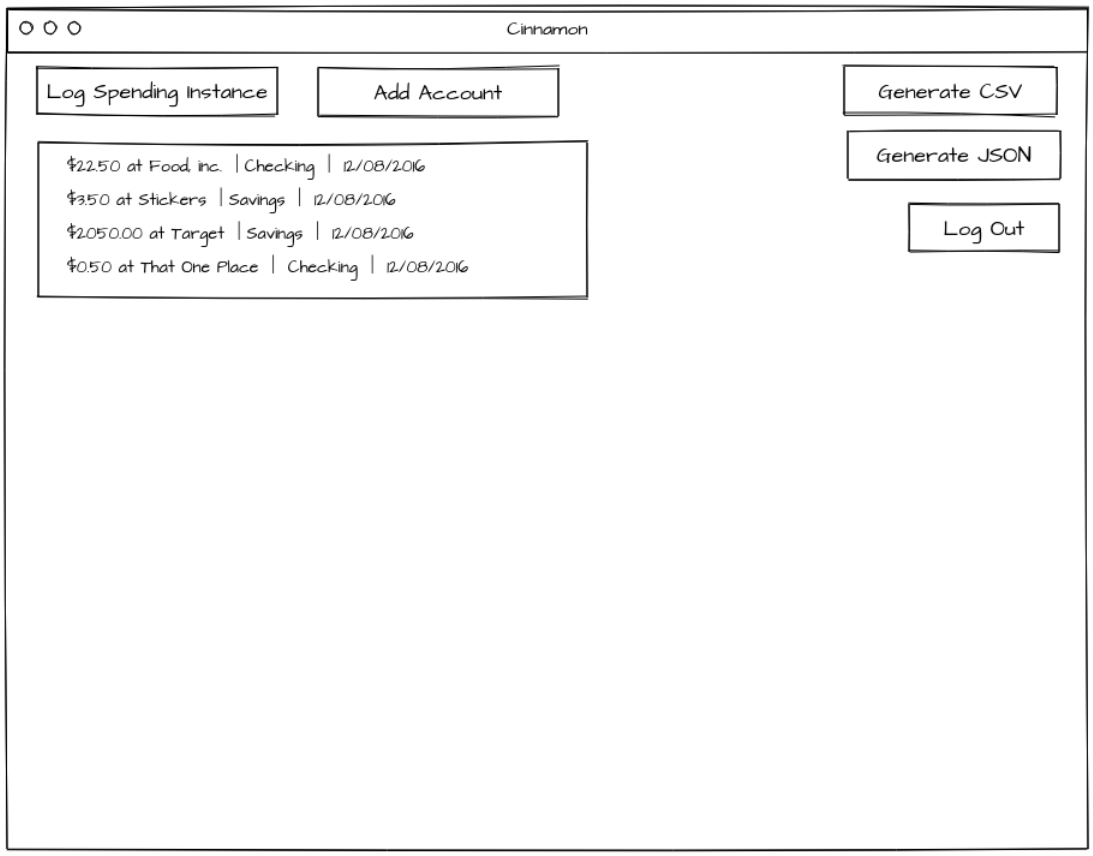


Figure 3: UI Mockup for home page. A main view to show spending instances and their details (amount, location. Also includes buttons to log spending instances, generate reports, add accounts, and log out of the application.

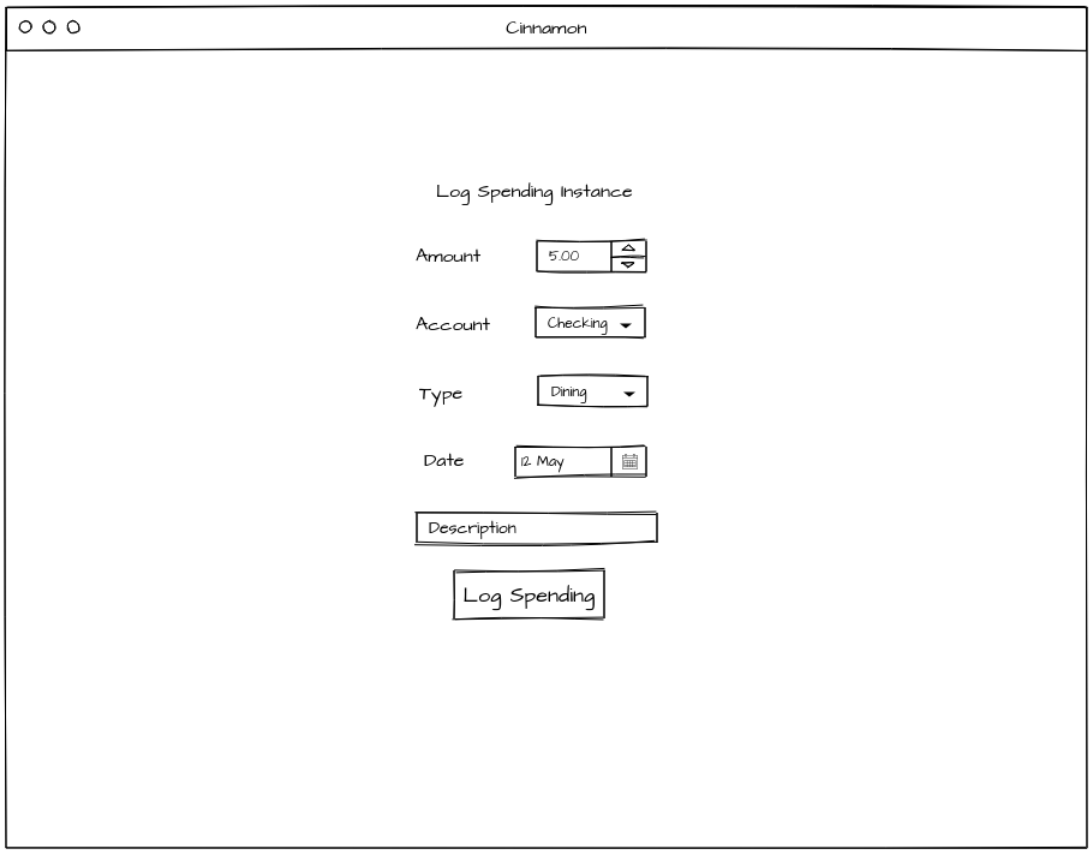


Figure 4: UI Mockup for log spending page. Form inputs for a spending instace. Options for accounts will be from the user's set accounts.

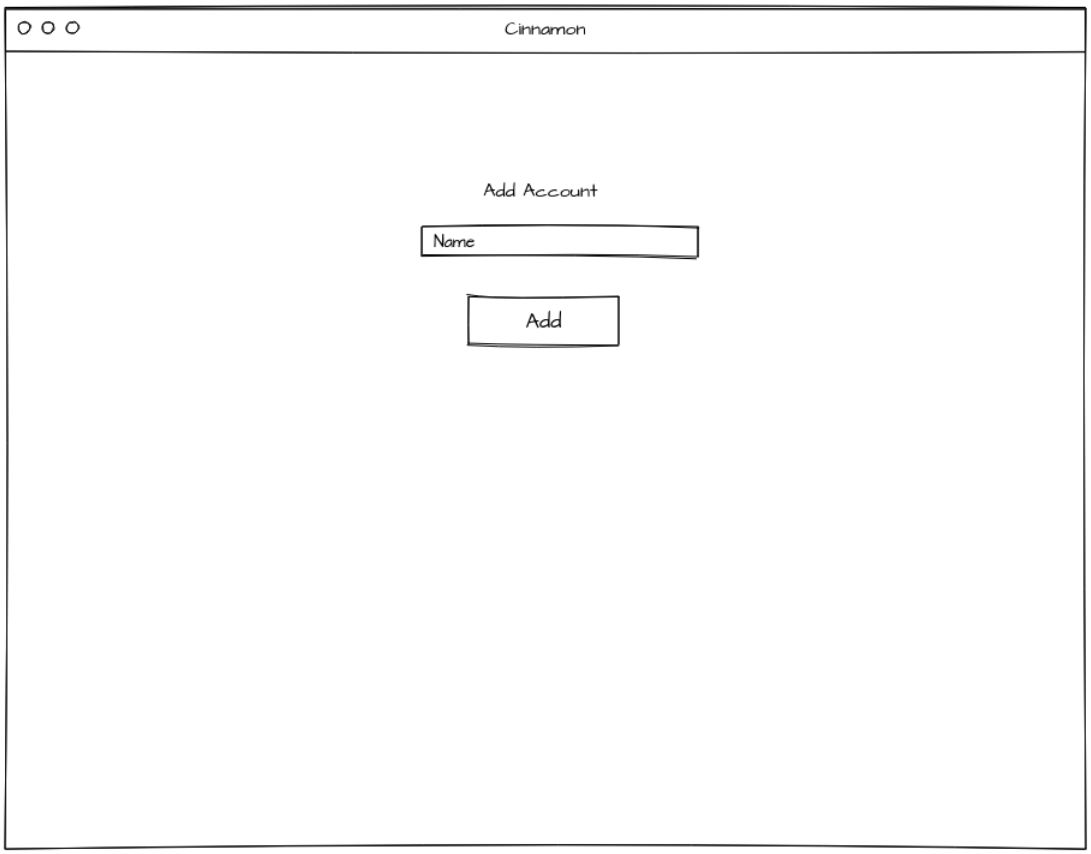
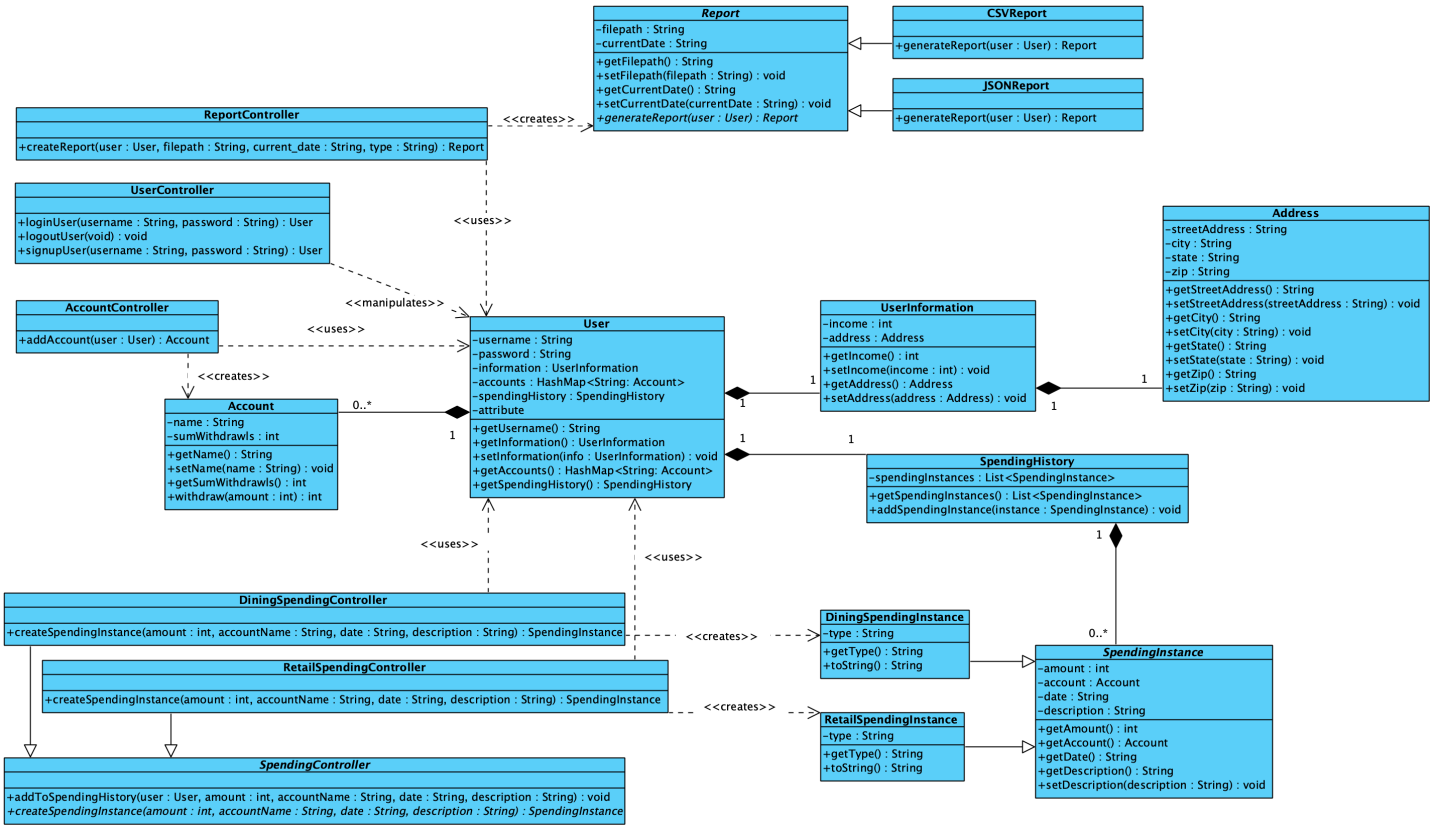


Figure 5: UI Mockup for add account page. Form input to add a new account.


Report
-filepath : String
-currentDate : String
+getFilepath() : String
+setFilepath(filepath : String) : void
+getCurrentDate() : String
+setCurrentDate(currentDate : String) : void
+generateReport(user : User) : Report
CSVReport
+generateReport(user : User) : Report
JSONReport
+generateReport(user : User) : Report
ReportController
+createReport(user : User, filepath : String, current_date : String, type : String) : Report
<<creates>>
UserController
+loginUser(username : String, password : String) : User
+logoutUser(void) : void
+signupUser(username : String, password : String) : User
<<uses>>
<<manipulates>>
AccountController
+addAccount(user : User) : Account
<<uses>>
<<creates>>
Account
-name : String
-sumWithdrawls : int
+getName() : String
+setName(name : String) : void
+getSumWithdrawls() : int
+withdraw(amount : int) : int
0..*
1
User
-username : String
-password : String
-information : UserInformation
-accounts : HashMap<String: Account>
-spendingHistory : SpendingHistory
-attribute
+getUsername() : String
+getInformation() : UserInformation
+setInformation(info : UserInformation) : void
+getAccounts() : HashMap<String: Account>
+getSpendingHistory() : SpendingHistory
1
1
UserInformation
-income : int
-address : Address
+getIncome() : int
+setIncome(income : int) : void
+getAddress() : Address
+setAddress(address : Address) : void
1
1
Address
-streetAddress : String
-city : String
-state : String
-zip : String
+getStreetAddress() : String
+setStreetAddress(streetAddress : String) : void
+getCity() : String
+setCity(city : String) : void
+getState() : String
+setState(state : String) : void
+getZip() : String
+setZip(zip : String) : void
1
1
SpendingHistory
-spendingInstances : List<SpendingInstance>
+getSpendingInstances() : List<SpendingInstance>
+addSpendingInstance(instance : SpendingInstance) : void
1
0..*
<<uses>>
<<uses>>
DiningSpendingController
+createSpendingInstance(amount : int, accountName : String, date : String, description : String) : SpendingInstance
RetailSpendingController
+createSpendingInstance(amount : int, accountName : String, date : String, description : String) : SpendingInstance
<<creates>>
<<creates>>
DiningSpendingInstance
-type : String
+getType() : String
+toString() : String
RetailSpendingInstance
-type : String
+getType() : String
+toString() : String
SpendingInstance
-amount : int
-account : Account
-date : String
-description : String
+getAmount() : int
+getAccount() : Account
+getDate() : String
+getDescription() : String
+setDescription(description : String) : void
SpendingController
+addToSpendingHistory(user : User, amount : int, accountName : String, date : String, description : String) : void
+createSpendingInstance(amount : int, accountName : String, date : String, description : String) : SpendingInstance