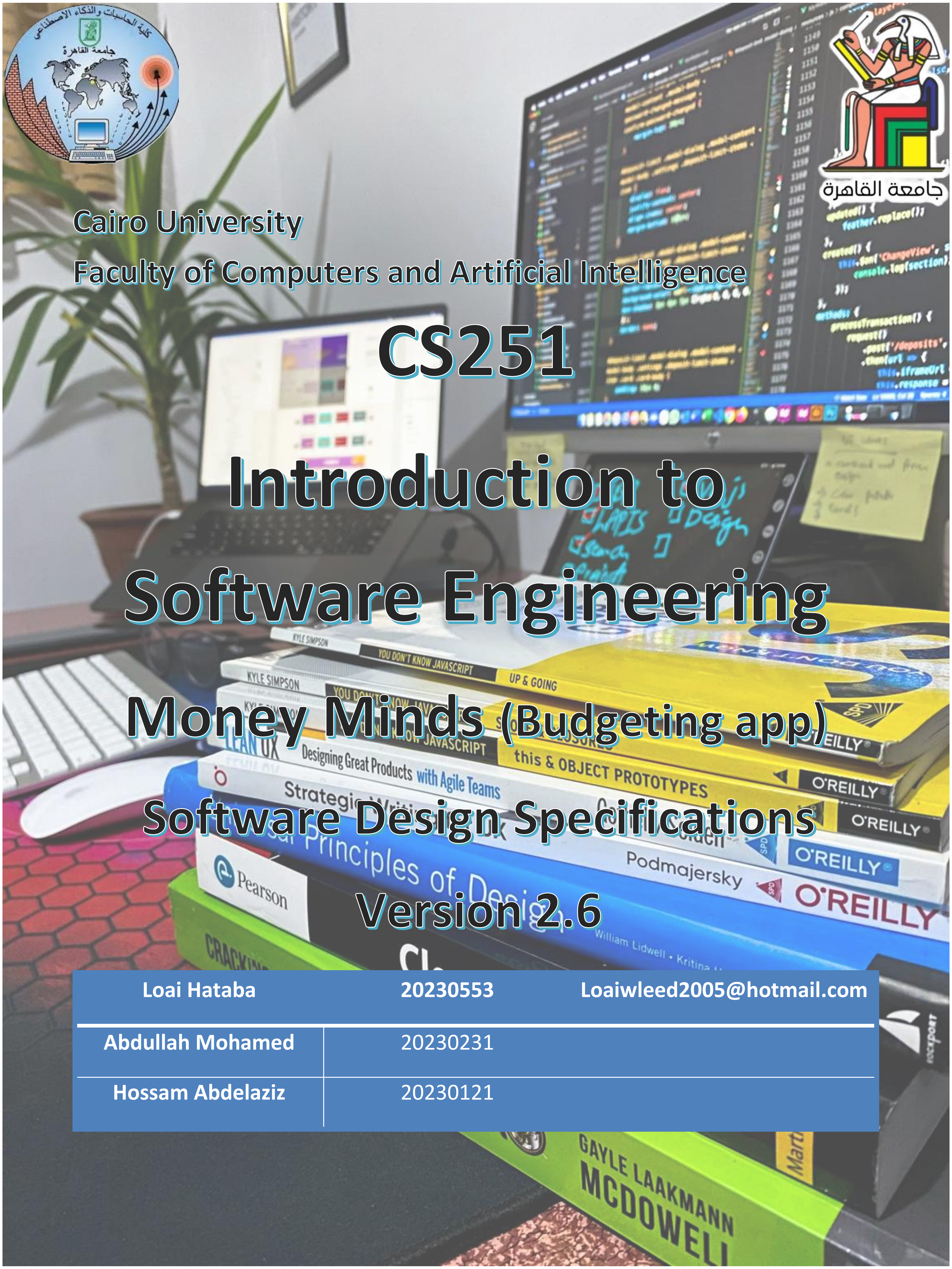


Cairo University

Faculty of Computers and Artificial Intelligence

CS251

Introduction to Software Engineering

Money Minds (Budgeting app)

Software Design Specifications

Version 2.6

Loai Hataba	**20230553**	**Loaiwleed2005@hotmail.com**
Abdullah Mohamed	20230231	
Hossam Abdelaziz	20230121	

CS251: HoodRatz
Project: Money Minds

Software Design Specification

Contents

Team 3
Document Purpose and Audience 3
System Models 4
I. Architecture Diagram 4
II. Class Diagram(s) 6
III. Class Descriptions 7
IV. Sequence diagrams 9
Class - Sequence Usage Table 11
V. State Diagram 11
VI. SOLID Principles 11
VII. Design Patterns 11
Tools 12
Ownership Report 12

CS251: HoodRatz
Project: Money Minds

Software Design Specification

Team

ID	Name	Email
20230553	Loai Hataba	20230553@stud.fci-cu.edu.eg
20230231	Abdullah Mohamed	20230231@stud.fci-cu.edu.eg
20230121	Hossam Abdelaziz	20230121@stud.fci-cu.edu.eg

Document Purpose and Audience

Purpose

• This document describes the design, structure, & functionality of the Budget Manager application.
• It explains how users can track their incomes, expenses, and generate financial reports.
• It outlines the main components, their responsibilities, and how they interact with each other.

Audience

- Developers – to understand the system architecture and build the application.
- Project Manager – to oversee the project development and ensure requirements are met.
- Testers/QA Team – to reference expected functionalities during testing.
- Potential Stakeholders (optional) – to review the overall app structure and features.

CS251: HoodRatz
Project: Money Minds

Software Design Specification

System Models

I. Architecture Diagram

Software Architecture Choice

For the Budget Manager application, we selected an **architecture** consisting of the **Frontend**, **Backend**, and **Database** layers, connected through APIs and supported by Authentication and Analytics services.
This architecture is suitable for the project because it provides:

- **Separation of concerns**: each layer has a specific responsibility (UI, business logic, data storage).
- **Scalability**: the application can grow by upgrading each tier independently.
- **Security**: user data can be protected through centralized authentication mechanisms.
- **Maintainability**: the structure simplifies debugging, updates, and future enhancements.

System Components

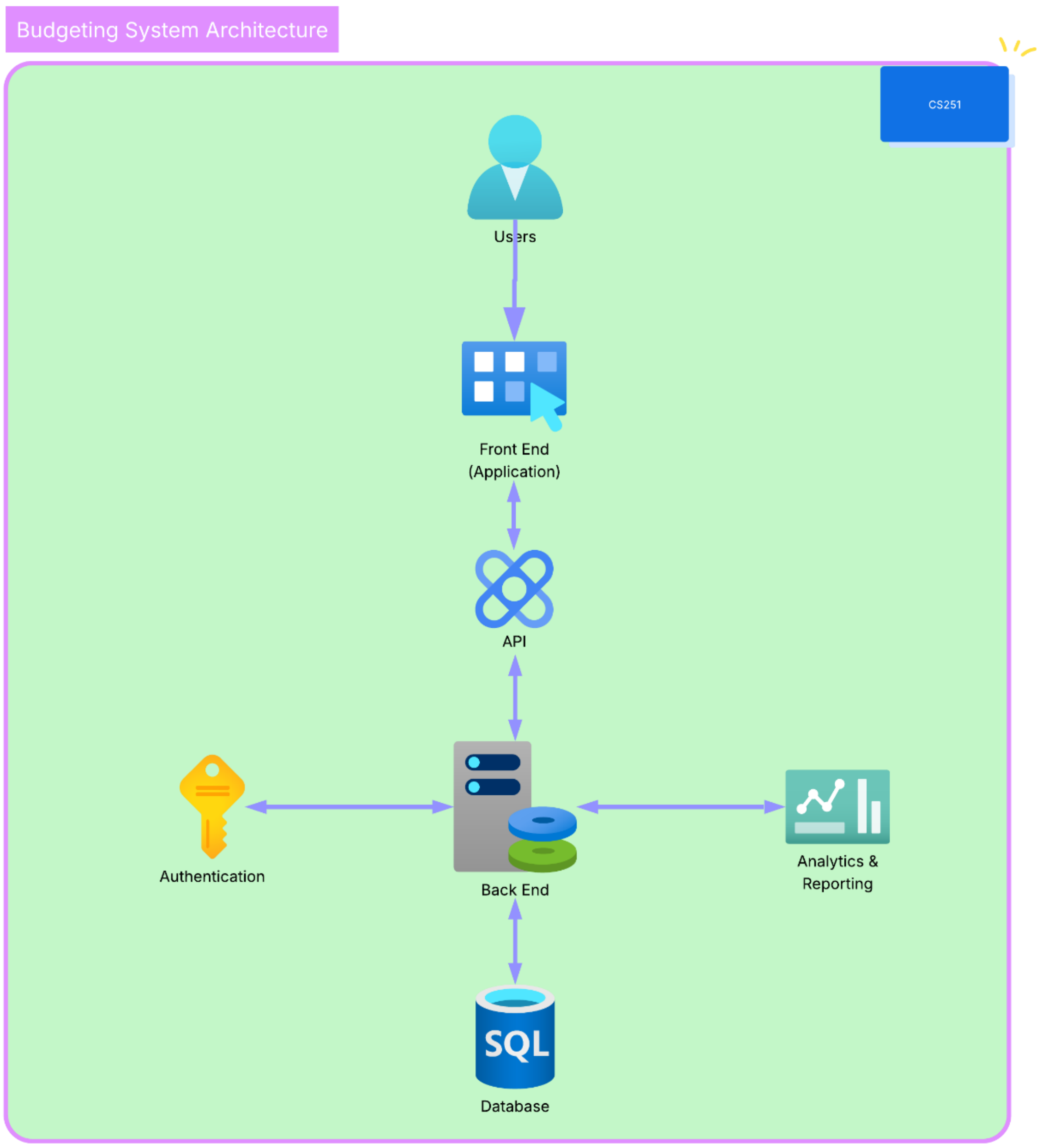
The system is divided into the following main components:

- **Users**: Individuals who interact with the application to manage their budgets.
- **Front End (Application)**: The graphical user interface that users interact with. It sends and receives data via APIs.
- **API**: Facilitates communication between the Front End and the Back End.
- **Back End**: Processes requests, applies business logic, manages authentication, reporting, and communicates with the database.
- **Authentication Service**: Handles user login, registration, and secure access management.
- **Database (SQL)**: Stores persistent data, including users' incomes, expenses, and transaction history.
- **Analytics & Reporting**: Generates financial reports and visual insights based on user data.

Software Design Specification

Architecture Diagram

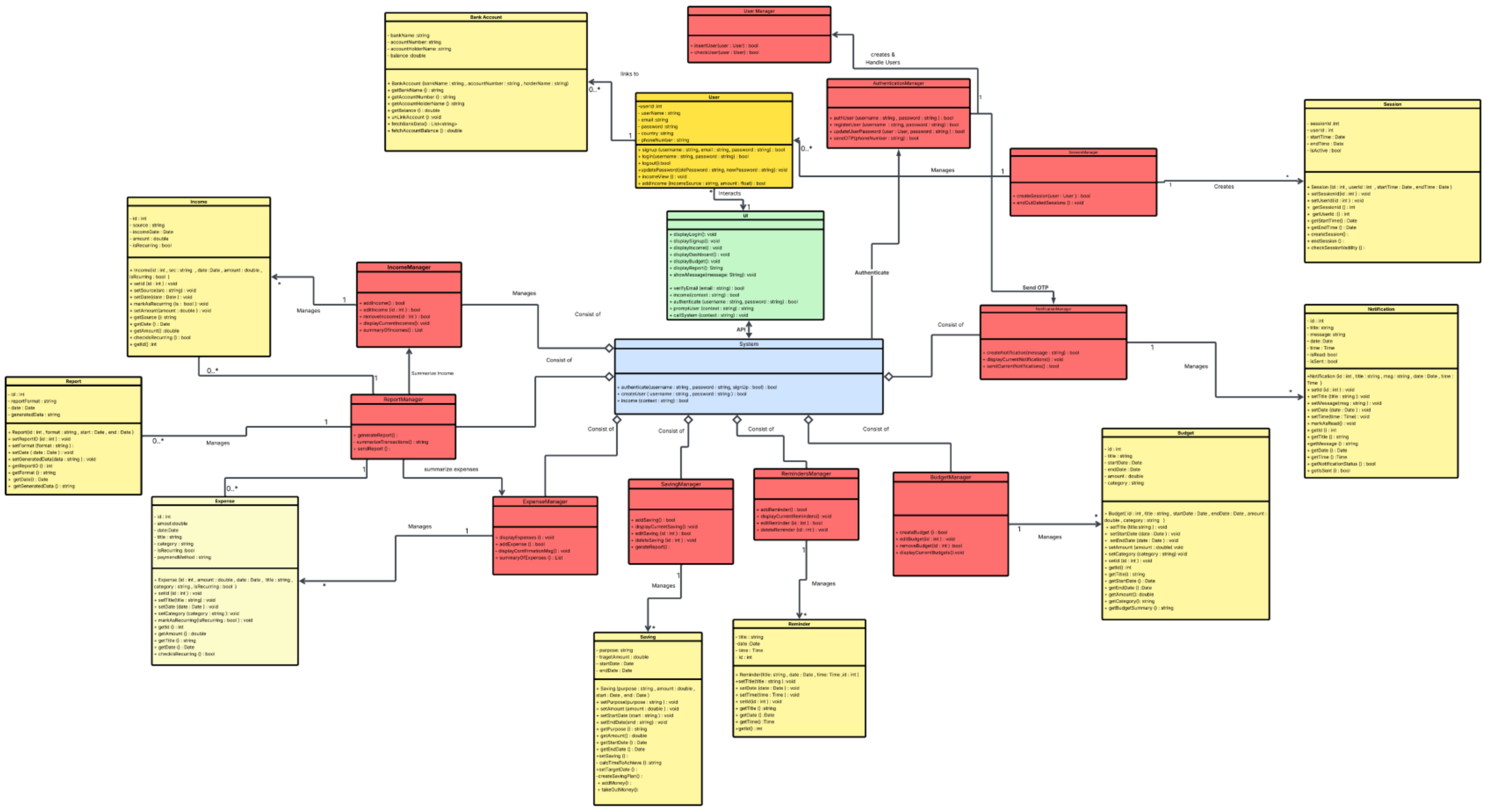
The architecture diagram below shows the relationship between different components using a simple arrow-and-box notation:



CS251: HoodRatz
Project: Money Minds

Software Design Specification

II. Class Diagram(s)



Bank Account
- bankName :string
- accountNumber: string
- accountHolderName :string
- balance :double
+ BankAccount (bankName : string , accountNumber : string , holderName : string)
+ getBankName () : string
+ getAccountNumber () : string
+ getAccountHolderName () :string
+ getBalance () : double
+ unLinkAccount () :void
+ fetchBankData() : List<string>
+ fetchAccountBalance () : double
links to
0..*
1
User Manager
+ insertUser(user : User) : bool
+ checkUser(user : User) : bool
creates & Handle Users
1
User
-userId :int
- userName : string
- email :string
- password :string
- country :string
- phoneNumber : string
+ signup (username : string, email : string, password : string) : bool
+ login(username : string, password : string) : bool
+ logout():bool
+updatePassword(oldPassword : string, newPassword : string): void
+ incomeView () : void
+ addIncome (incomeSource : string, amount : float) : bool
AuthenticationManager
+ authUser (username : string , password : string) : bool
+ registerUser (username : string, password : string) : bool
+ updateUserPassword (user : User, password : string) : bool
+ sendOTP(phoneNumber : string) : bool
0..*
Manages
1
SessionManager
+ createSession(user : User) : bool
+ endOutDatedSessions () : void
1
Creates
*
Session
- sessionId :int
- userId : int
- startTime : Date
- endTime : Date
- isActive : bool
+ Session (id : int , userId : int , startTime : Date , endTime : Date)
+ setSessionId(id : int) : void
+ setUserId(id : int) : void
+ getSessionId () : int
+ getUserId : () : int
+ getStartTime() : Date
+ getEndTime () : Date
+ createSession() :
+ endSession () :
+ checkSessionValidity () :
*
Interacts
1
UI
+ displayLogin(): void
+ displaySignup(): void
+ displayIncome() : void
+ displayDashboard() : void
+ displayBudget(): void
+ displayReport(): String
+ showMessage(message: String): void
+ verifyEmail (email : string) : bool
+ income(context : string) : bool
+ authenticate (username : string, password : string) : bool
+ promptUser (context : string) : string
+ callSystem (context : string) : void
Authenticate
Send OTP
API
Income
- id : int
- source : string
- incomeDate : Date
- amount : double
- isRecurring : bool
+ Income(id : int , src : string , date :Date , amount : double , isRcurring : bool)
+ setId (id : int) : void
+ setSource(src : string) : void
+ setDate(date : Date) : void
+ markAsRecurring (is : bool): void
+ setAmount(amount : double) : void
+ getSource (): string
+ getDate () : Date
+ getAmount() :double
+ checkIsRecurring () : bool
+ getId() :int
*
Manages
1
IncomeManager
+ addIncome() : bool
+ editIncome (id : int) : bool
+ removeIncome(id : int) : bool
+ displayCurrentIncomes(): void
+ summaryOfIncomes() : List
Manages
Consist of
Consist of
System
+ authenticate(username : string , password : string, signUp : bool) : bool
+ createUser (username : string , password : string) : bool
+ income (context : string) : bool
Consist of
NotificationManager
+ createNotification(message : string) : bool
+ displayCurrentNotifications() : void
+ sendCurrentNotifications() : bool
1
Manages
*
Notification
- id : int
- title: string
- message: string
- date: Date
- time : Time
- isRead: bool
- isSent : bool
+Notification (id : int , title : string , msg : string , date : Date , time : Time)
+ setId (id : int) : void
+ setTitle (title : string): void
+ setMessage(msg : string) : void
+ setDate (date : Date) : void
+ setTime(time : Time) : void
+ markAsRead() : void
+ getId () : int
+ getTitle () : string
+getMessage () : string
+ getDate () : Date
+ getTime () :Time
+ getNotificationStatus () : bool
+ getIsSent () : bool
0..*
1
Summarize Income
Report
- id : int
- reportFormat : string
- date : Date
- generatedData : string
+ Report(id : int , format : string , start : Date , end : Date)
+ setReportID (id : int) : void
+ setFormat (format : string) :
+ setDate (date : Date) : void
+ setGeneratedData(data : string) : void
+ getReportID () : int
+ getFormat () : string
+ getDate() : Date
+ getGeneratedData () : string
0..*
Manages
1
ReportManager
+ generateReport() :
- summarizeTransactions() : string
+ sendReport () :
1
0..*
summarize expenses
Expense
- id : int
- amout:double
- date:Date
- title : string
- category : string
- isRecurring :bool
- paymendMethod : string
+ Expense (id : int , amount : double , date : Date , title : string , category : string , isRecurring : bool)
+ setId (id : int) : void
+ setTitle(title : string) : void
+ setDate (date : Date) : void
+ setCategory (category : string): void
+ markAsRecurring(isRecurring : bool) : void
+ getId () : int
+ getAmount () : double
+ getTitle () : string
+ getDate () : Date
+ checkIsRecurring () : bool
*
Manages
1
ExpenseManager
+ displayExpenses () : void
+ addExpense () : bool
- displayConfirmationMsg() : void
+ summaryOfExpenses () : List
Consist of
Consist of
Consist of
Consist of
SavingManager
+ addSaving() : bool
+ displayCurrentSaving() :void
+ editSaving (id : int) : bool
+ deleteSaving (id : int) : void
+ gerateReport() :
1
Manages
*
Saving
- purpose: string
- tragetAmount : double
- startDate : Date
- endDate : Date
+ Saving (purpose : string , amount : double , start : Date , end : Date)
+ setPurpose(purpose : string) : void
+ setAmount (amount : double) : void
+ setStartDate (start : string) : void
+ setEndDate(end : string) : void
+ getPurpose () : string
+ getAmount() : double
+ getStartDate () : Date
+ getEndDate () : Date
+setSaving () :
- calcTimeToAchieve () :string
+setTargetDate () :
-createSavingPlan() :
+ addMoney() :
+ takeOutMoney():
RemindersManager
+ addReminder() : bool
+ displayCurrentReminders() :void
+ editReminder (id : int) : bool
+ deleteReminder (id : int) : void
1
Manages
*
Reminder
- title : string
-date :Date
- time : Time
- id : int
+ Reminder(title: string , date : Date , time: Time ,id : int)
+setTitle(title : string) :void
+ setDate (date : Date) : void
+ setTime(time : Time) : void
+ setId(id : int) : void
+ getTitle () :string
+ getDate () :Date
+ getTime() :Time
+getId() : int
BudgetManager
+ createBudget () : bool
+ editBudget(id : int) : void
+ removeBudget(id : int) :bool
+ displayCurrentBudgets():void
1
Manages
*
Budget
- id : int
- title : string
- startDate : Date
- endDate : Date
- amount : double
- category : string
+ Budget(id : int , title : string , startDate : Date , endDate : Date , amount : double , category : string)
+ setTitle (title:string) : void
+ setStartDate (date : Date) : void
+ setEndDate (date : Date) : void
+ setAmount (amount : double) void
+ setCategory (category : string) void
+ setId (id : int) : void
+ getId() :int
+ getTitle() : string
+ getStartDate () : Date
+ getEndDate () :Date
+ getAmount(): double
+ getCategory(): string
+ getBudgetSummary () : string

Software Design Specification

III. Class Descriptions

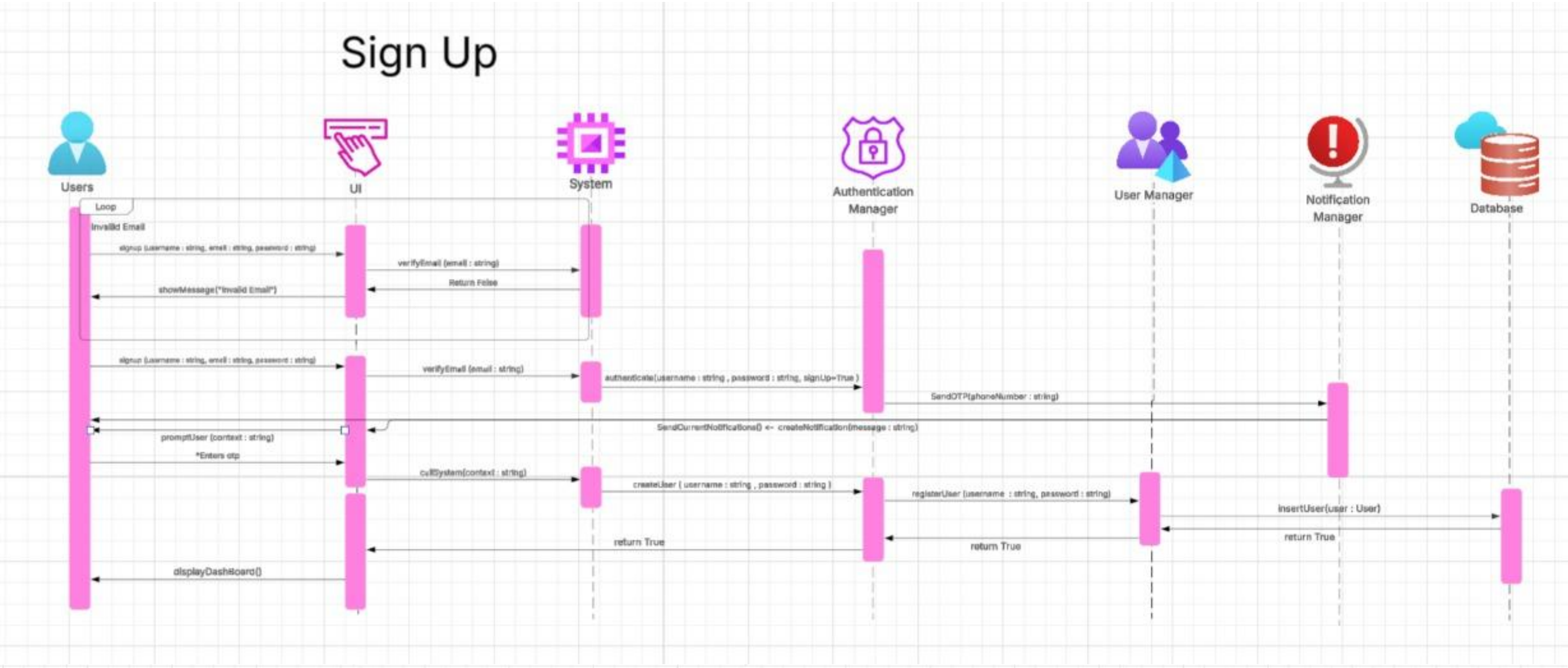
Class ID	Class Name	Description & Responsibility
1	Income	Represents an income entry with properties like source, amount, and date; responsible for managing income-related operations.
2	IncomeManager	Manages multiple Income objects; responsible for adding, deleting, retrieving, and summarizing incomes.
3	BankAccount	Represents a user's bank account details; responsible for storing account number, balance, and bank name.
4	Report	Represents financial reports; responsible for summarizing income and expenses over a time period.
5	ReportManager	Manages creation and retrieval of financial reports based on user data.
6	Expense	Represents an expense entry with properties like type, amount, and description; manages individual expense records.
7	ExpenseManager	Manages multiple Expense objects; responsible for adding, deleting, and retrieving expenses.
8	Saving	Represents a saving goal or entry; manages target amounts and current savings status.
9	SavingManager	Manages user savings; responsible for adding savings and generating saving reports.
10	User	Represents a system user with authentication credentials; manages personal user details.
11	UserManager	Manages creating and checking for users in the database.
12	Budget	Represents a budget plan for a category or time period; manages allocation and spending tracking.
13	BudgetManager	Manages user budgets; responsible for creating and managing budget plans.
14	Notification	*Represents a message or alert sent to users; responsible for delivering real-time updates, reminders, or warnings based on system events or user actions.*
15	Notification Manager	Represents a notification message; manages sending alerts to users.
16	AuthenticationManager	Responsible for verifying and managing user authentication (login/signup).
17	Reminder	*Represents a scheduled alert for important financial activities or goals; responsible for setting, updating, and managing reminders triggered at specific times or conditions.*
18	Reminder Manager	Represents a reminder entity; manages notification scheduling.
19	UI	Represents the front end of the application where the user would interact with the system.

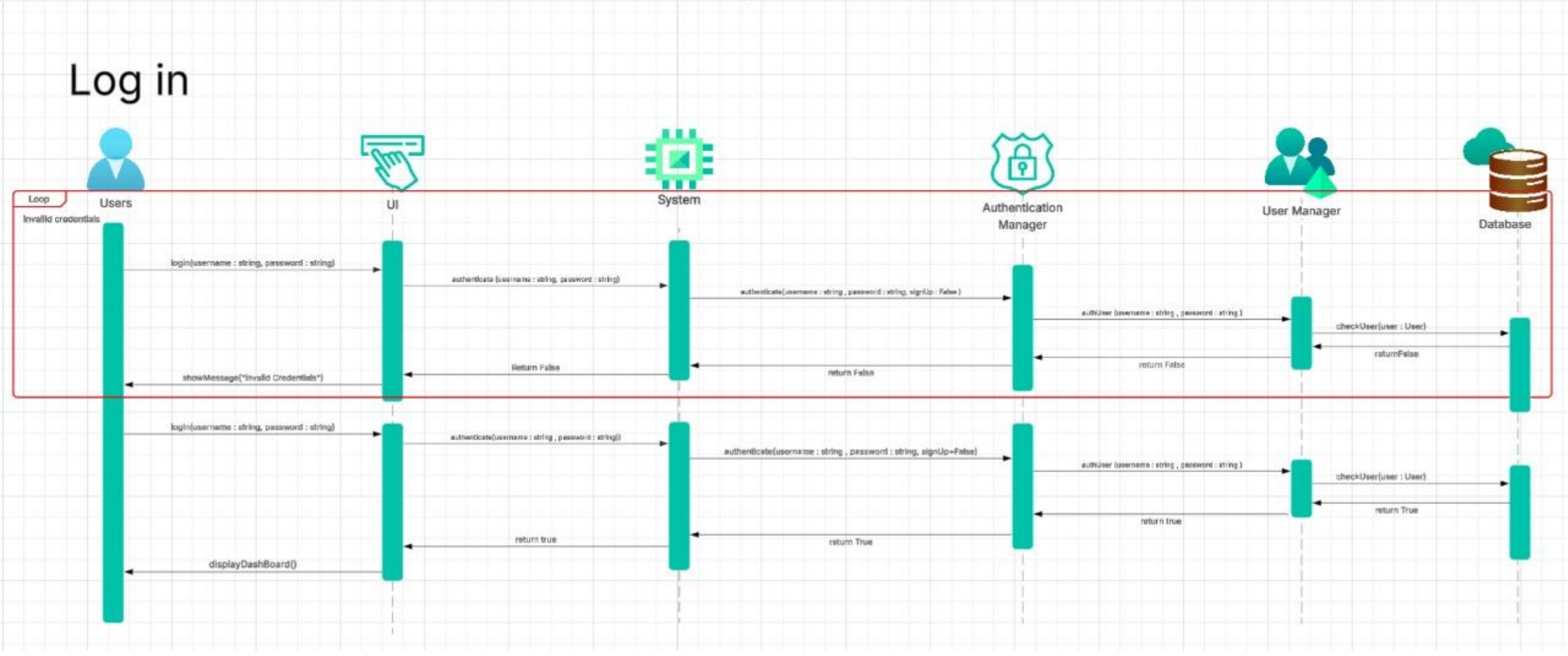
Software Design Specification

Class ID	Class Name	Description & Responsibility
20	System	Central class represents the entire system; that coordinates between managers and entities.
21	Session	Represents a user's active interaction period with the system; responsible for temporarily storing user data (such as login state…) during usage, until the session ends or expires.
22	Session Manager	*Responsible for creating, maintaining, and terminating user sessions; manages session-related data like active users, timeouts, and session validation to ensure continuous and secure user interaction.*

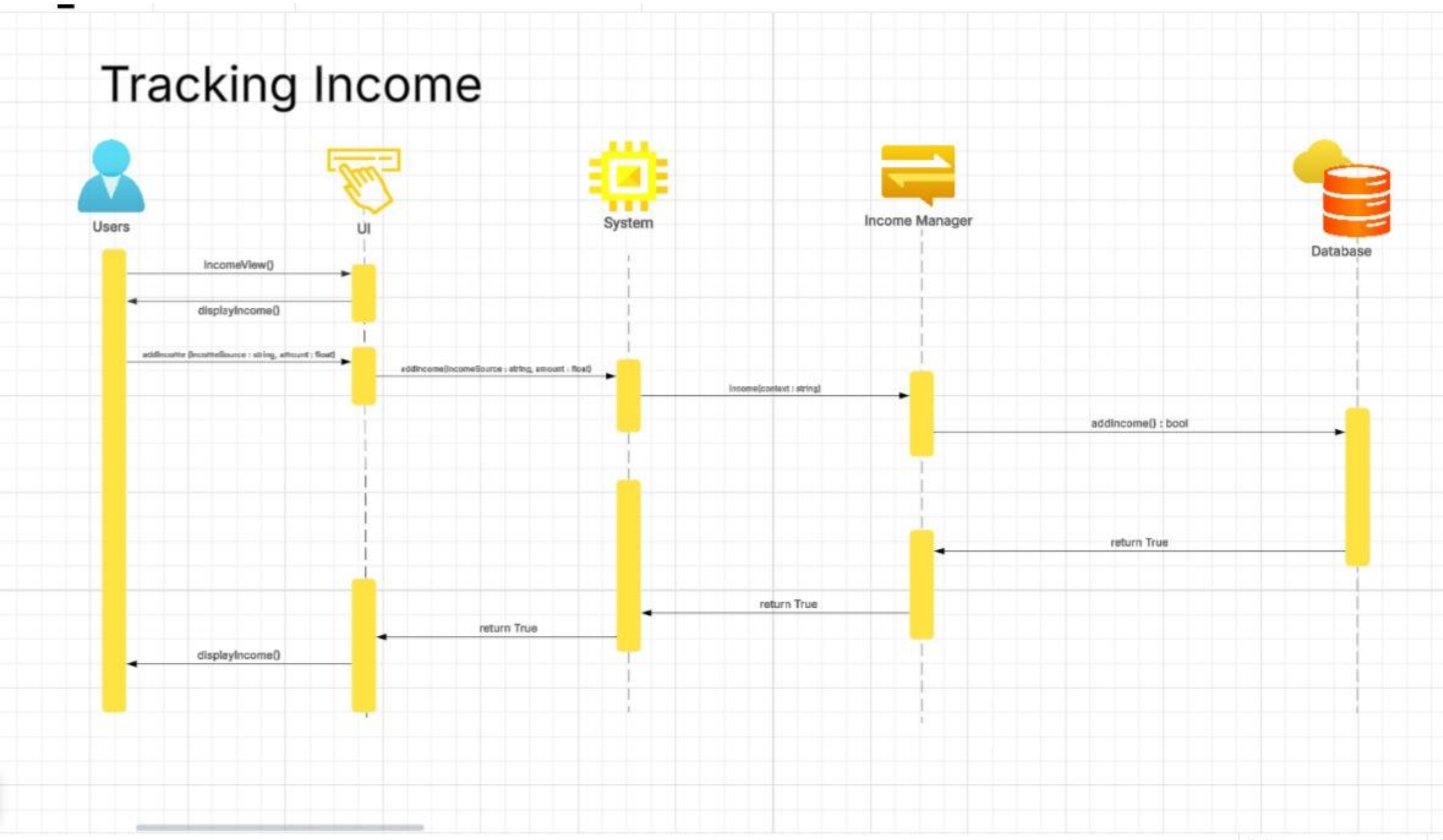
Software Design Specification

IV. Sequence diagrams


Sign Up
Users
UI
System
Authentication Manager
User Manager
Notification Manager
Database
Loop
Invalid Email
signup (username : string, email : string, password : string)
verifyEmail (email : string)
Return False
showMessage("Invalid Email")
signup (username : string, email : string, password : string)
verifyEmail (email : string)
authenticate(username : string , password : string, signUp=True)
SendOTP(phoneNumber : string)
SendCurrentNotifications() <- createNotification(message : string)
promptUser (context : string)
*Enters otp
callSystem(context : string)
createUser (username : string , password : string)
registerUser (username : string, password : string)
insertUser(user : User)
return True
return True
return True
displayDashBoard()


Log in
Users
UI
System
Authentication Manager
User Manager
Database
Loop
Invalid credentials
login(username : string, password : string)
authenticate (username : string, password : string)
authenticate(username : string , password : string, signUp : False)
authUser (username : string , password : string)
checkUser(user : User)
returnFalse
return False
return False
Return False
showMessage("Invalid Credentials")
login(username : string, password : string)
authenticate(username : string , password : string))
authenticate(username : string , password : string, signUp=False)
authUser (username : string , password : string)
checkUser(user : User)
return True
return true
return True
return true
displayDashBoard()

Software Design Specification



CS251: HoodRatz
Project: Money Minds

Software Design Specification

Class - Sequence Usage Table

Sequence Diagram	Classes Used	All Methods Used

1. Sign Up	Users
	UI
	System
	Authentication Manager
	User Manager
	Notification Manager
		signup (username : string, email : string, password : string)
		showMessage("Invalid Email")
		verifyEmail (email : string)
		authenticate(username : string , password : string, signUp : bool)
		SendOTP(phoneNumber : string)
		SendCurrentNotifications() createNotification(message : string)
		promptUser (context : string)
		callSystem(context : string)
		createUser (username : string , password : string)
		registerUser (username : string, password : string)
		insertUser(user : User)
		displayDashBoard()

2. Log in	Users
	UI
	System
	Authentication Manager
	User Manager
	Notification Manager
		login(username : string, password : string)
		authenticate (username : string, password : string)
		authUser (username : string , password : string)
		checkUser(user : User)
		showMessage("Invalid Credentials")
		displayDashBoard()

3. Track Income	Users
	UI
	System
	Income Manager
		incomeView()
		displayIncome()
		addIncome (incomeSource : string, amount : float)
		income(context : string)
		addIncome() : bool

V. State Diagram

- **For the <u>ONE MOST IMPORTANT</u> object, draw a state diagram to show the developer the different states it can be in. (for example it is initially created, then it can be shipped, cancelled (if cancelling is possible), …., etc.)**

VI. SOLID Principles

- **Explain how you applied <u>THREE OF THE SOLID PRINCIPLES</u> in your design and show the part that the principles where applied in.**

VII. Design Patterns

- **Use at least <u>THREE DESIGN PATTERNS,</u> any ones from the 23 patterns, not just ones explained in lecture. Explain where you used it and what was the benefit of using it in this place.**

CS251: HoodRatz
Project: Money Minds

Software Design Specification

Tools

- LucidChart

Ownership Report

Item	Owners
Loai Hataba	System Architecture & Sequence Diagrams
Abdullah Mohammed	
Hossam Abdelaziz	