1.Problem Domain: Logitech

Description: The problem domain revolves around the challenges faced by individuals in managing their finances effectively and making informed investment decisions. It encompasses various aspects of personal finance, including budgeting, saving, investing, and retirement planning. The problem domain aims to address the following issues:

1. Lack of Financial Knowledge: Many individuals have limited knowledge and understanding of financial concepts, investment strategies, and long-term financial planning. This lack of knowledge hinders their ability to make informed decisions and maximize their financial growth.

2. Limited Access to Investment Opportunities: People often face difficulties in accessing a wide range of investment options, especially those that offer a balance between safety, income, and growth. Limited access to diverse investment opportunities restricts their ability to diversify their portfolios and potentially achieve higher returns.

3. Complex Financial Processes: The financial processes involved in managing investments, such as account opening, fund transfers, investment tracking, and monitoring performance, can be complex and time-consuming. The lack of user-friendly tools and streamlined processes makes it challenging for individuals to navigate the financial landscape efficiently.

4. Risk Management: Understanding and managing investment risks is crucial for long-term financial success. However, many individuals struggle to assess and mitigate risks effectively, leading to potential financial losses and missed opportunities.

5. Retirement Planning: Planning for retirement is a significant financial challenge for many individuals. They often lack the knowledge and tools to estimate their future financial needs, set retirement goals, and create effective investment strategies to achieve those goals.

2.1 Active Actors

1. User: Represents individuals who use the logitech application. They can create accounts, explore investment programs, make transactions, and manage their financial goals.

2. Financial Management System: Represents the backend system that powers the logitech application. It includes the logic and functionality to create and manage user accounts, investment accounts, investment programs, and investment transactions.

3. Investment Account: Represents an account held by a user within the logitech application. It tracks the account type, balance, start and end dates, and is associated with a specific user.

4. Investment Program: Represents various investment opportunities available to users. It includes details such as the program name, description, type, investment amounts, expected returns, and risk level.

5. Investment Transaction: Represents a financial transaction made by a user within the application. It includes details such as the user, account, program, transaction type, date, and amount.

2.3 Passive Actors

1. External Financial Institutions: These are external entities such as banks, investment firms, or financial institutions that may be integrated with the Logitech application. They can provide services like account verification, fund transfers, or accessing external investment programs.
2. Market Data Providers: These are external entities that supply market data, such as stock prices, currency exchange rates, or economic indicators. The Logitech application may integrate with these providers to fetch real-time or historical data for investment analysis and decision-making.
3. Notification Service: This is a passive actor that handles the delivery of notifications or alerts to users. It may send notifications related to account balances, investment program updates, transaction confirmations, or important financial information.
4. Regulatory Bodies: These are entities responsible for regulating financial activities and ensuring compliance with relevant laws and regulations. The Logitech application may need to adhere to regulatory requirements, and interactions with regulatory bodies can be considered as passive actors.
5. Reporting and Analytics Tools: These tools enable the generation of reports, analytics, and insights based on user data, investment performance, and financial trends. The Logitech application may utilize such tools to provide users with visualizations, performance summaries, or personalized recommendations.

2.2 Functions

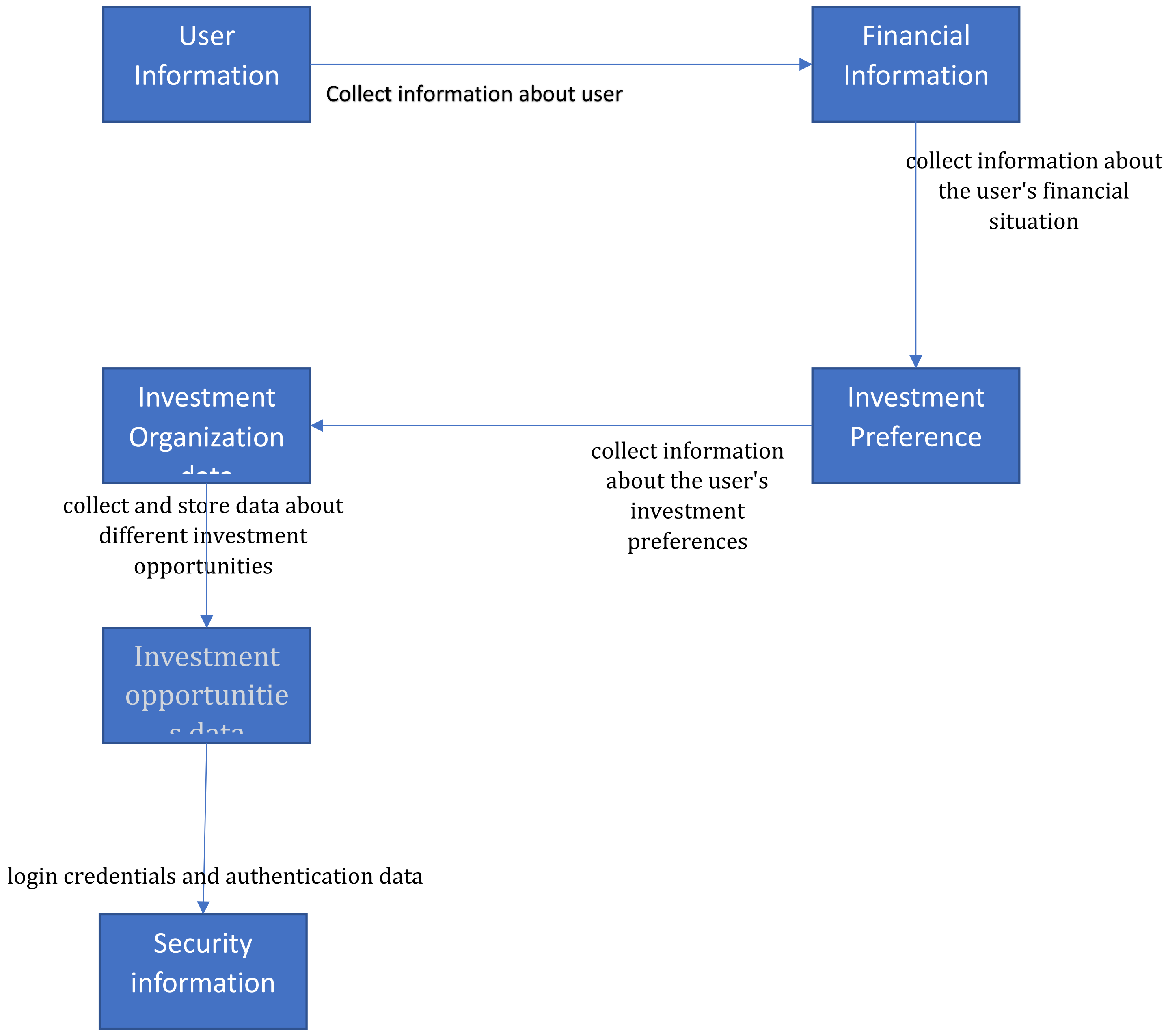
```
data class User(

   val userId: String,

   val userFname: String,

   val userLname: String,

   val name: String,

   val email: String,

)


data class InvestmentAccount(

   val accountId: String,

   val userId: String,

   val accountType: String,

   val accountBalance: Double,

   val startDate: Date,

   val endDate: Date

)
```

```
data class InvestmentProgram(
   val programId: String,
   val programName: String,
   val programDescription: String,
   val programType: String,
   val minInvestmentAmount: Double,
   val maxInvestmentAmount: Double,
   val expectedReturn: Double,
   val riskLevel: String
)

data class InvestmentTransaction(
   val transactionId: String,
   val userId: String,
   val accountId: String,
   val programId: String,
   val transactionType: String,
   val transactionDate: Date,
   val amount: Double
)

class FinancialManagementSystem {
   private val users: MutableList<User> = mutableListOf()
   private val investmentAccounts: MutableList<InvestmentAccount> = mutableListOf()
   private val investmentPrograms: MutableList<InvestmentProgram> = mutableListOf()
   private val investmentTransactions: MutableList<InvestmentTransaction> = mutableListOf()

   fun createUser(name: String, age: Int, email: String, contactNumber: String, address: String): User {
      val userId = generateUniqueId() // Function to generate a unique user ID
      val user = User(userId, name, age, email, contactNumber, address)
```

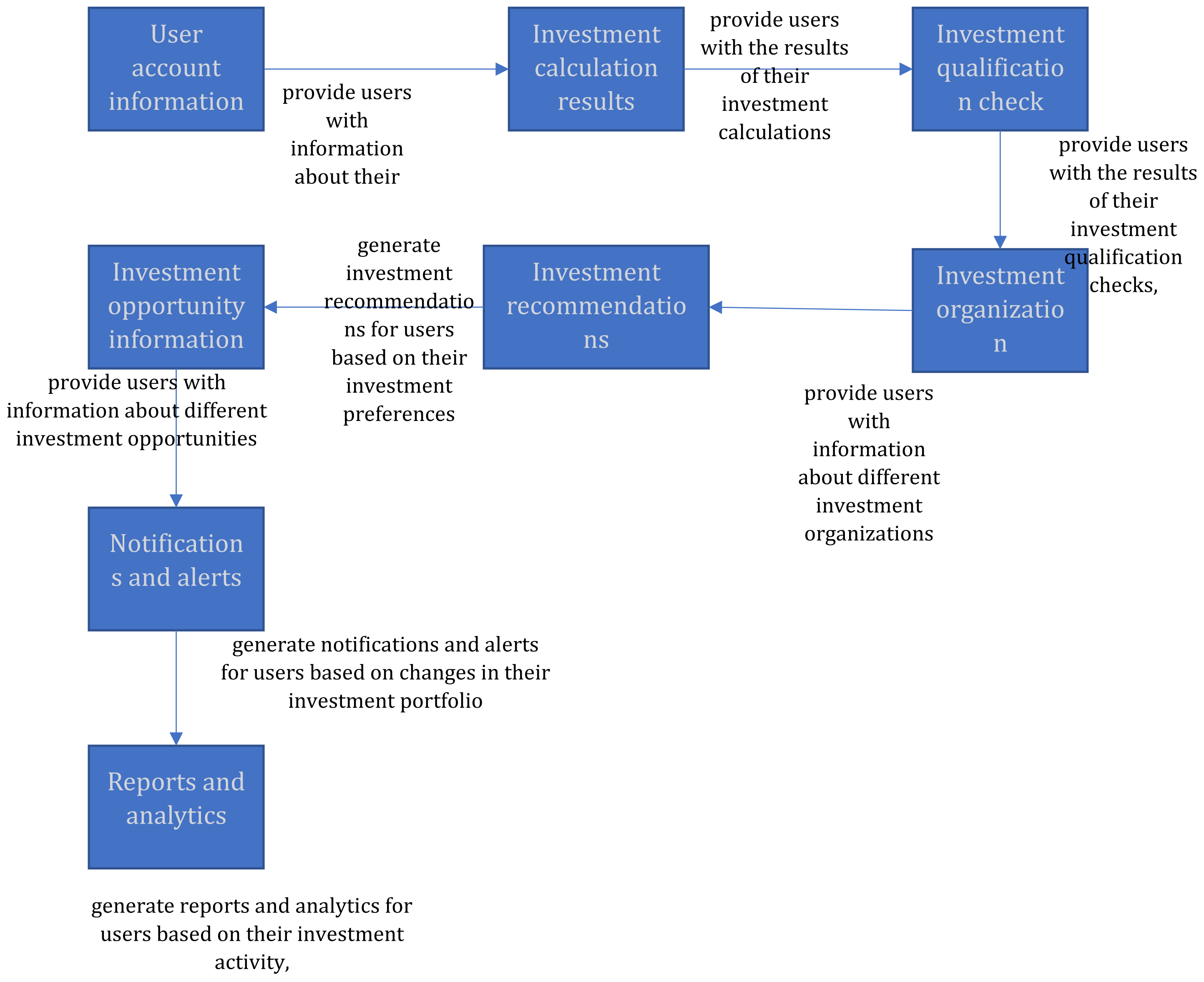
```
    users.add(user)
    return user
}

fun createInvestmentAccount(
    userId: String,
    accountType: String,
    accountBalance: Double,
    startDate: Date,
    endDate: Date
): InvestmentAccount {
    val accountId = generateUniqueId() // Function to generate a unique account ID
    val investmentAccount =
        InvestmentAccount(accountId, userId, accountType, accountBalance, startDate, endDate)
    investmentAccounts.add(investmentAccount)
    return investmentAccount
}

fun createInvestmentProgram(
    programName: String,
    programDescription: String,
    programType: String,
    minInvestmentAmount: Double,
    maxInvestmentAmount: Double,
    expectedReturn: Double,
    riskLevel: String
): InvestmentProgram {
    val programId = generateUniqueId() // Function to generate a unique program ID
    val investmentProgram = InvestmentProgram(
        programId,
        programName,
            programDescription,
            programType,
            minInvestmentAmount,
            maxInvestmentAmount,
            expectedReturn,
            riskLevel
        )
        investmentPrograms.add(investmentProgram)
        return investmentProgram
    }

    fun createInvestmentTransaction(
        userId: String,
        accountId: String,
        programId: String,
        transactionType: String,
        transactionDate: Date,
        amount: Double
    ): InvestmentTransaction {
        val transactionId = generateUniqueId() // Function to generate a unique transaction ID
        val investmentTransaction = InvestmentTransaction(
            transactionId,
            userId,
            accountId,
            programId,
            transactionType,
            transactionDate,
            amount
        )
        investmentTransactions.add(investmentTransaction)
        return investmentTransaction
```

```
}

fun getUserById(userId: String): User? {
    return users.find { it.userId == userId }
}

fun getInvestmentAccountById(accountId: String): InvestmentAccount? {
    return investmentAccounts.find { it.accountId == accountId }
}

fun getInvestmentProgramById(programId: String): InvestmentProgram? {
    return investmentPrograms.find { it.programId == programId }
}

fun getInvestmentTransactionsByUserId(userId: String): List<InvestmentTransaction> {
    return investmentTransactions.filter { it.userId == userId }
}

fun getInvestmentTransactionsByAccountId(accountId: String): List<InvestmentTransaction> {
    return investmentTransactions.filter { it.accountId == accountId }
}

fun calculateAccountBalance(accountId: String): Double {
    var accountBalance = 0.0
    investmentTransactions.forEach { transaction ->
        if (transaction.accountId == accountId) {
            when (transaction.transactionType) {
                "Deposit" -> accountBalance += transaction.amount
                "Withdrawal" -> accountBalance -= transaction.amount
                "Investment" -> {
                    val program = getInvestmentProgramById(transaction.programId)
                    if (program != null) {
                        accountBalance += transaction.amount * program.expectedReturn
                    }
                }
            }
        }
    }
    return accountBalance
}

private fun generateUniqueId(): String {
    // Function to generate a unique ID
    return UUID.randomUUID().toString()
}
}
```

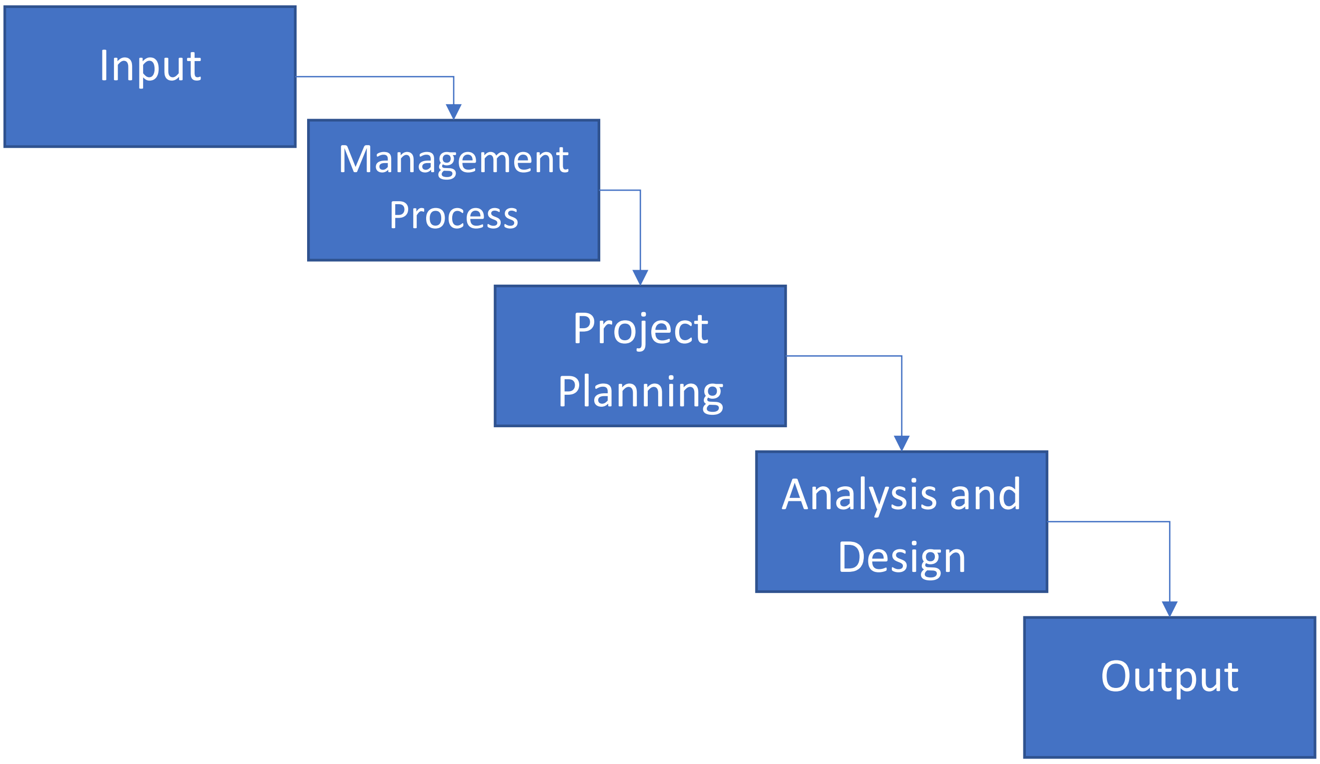
3.1 – Input Specifications


User Information
Collect information about user
Financial Information
collect information about the user's financial situation
Investment Preference
collect information about the user's investment preferences
Investment Organization data
collect and store data about different investment opportunities
Investment opportunities data
login credentials and authentication data
Security information

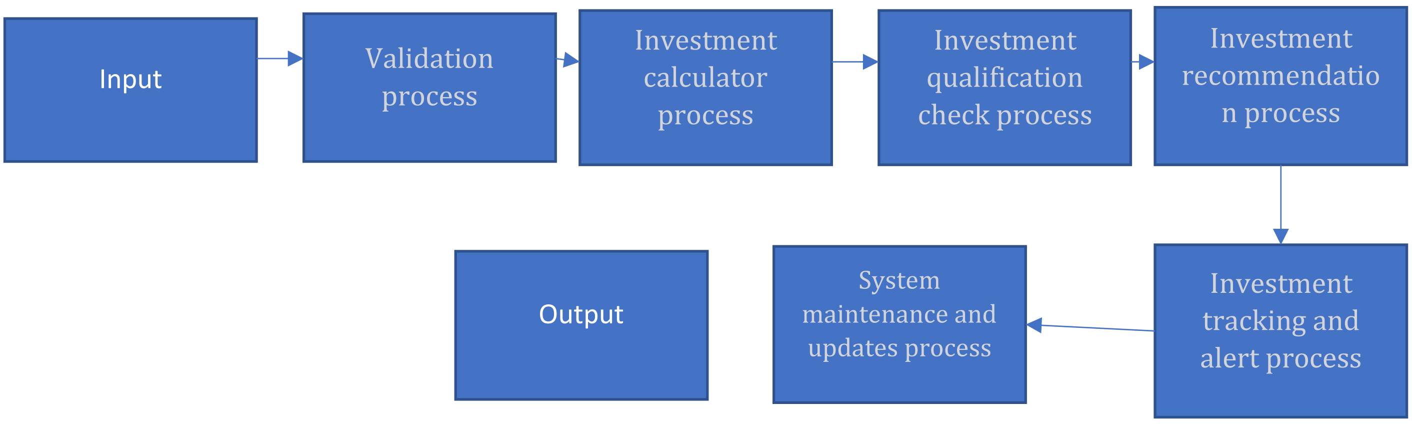
3.2 – Output Specifications


User account information
provide users with information about their
Investment calculation results
provide users with the results of their investment calculations
Investment qualification check
provide users with the results of their investment qualification checks,
Investment organization
provide users with information about different investment organizations
Investment recommendations
generate investment recommendations for users based on their investment preferences
Investment opportunity information
provide users with information about different investment opportunities
Notifications and alerts
generate notifications and alerts for users based on changes in their investment portfolio
Reports and analytics
generate reports and analytics for users based on their investment activity,

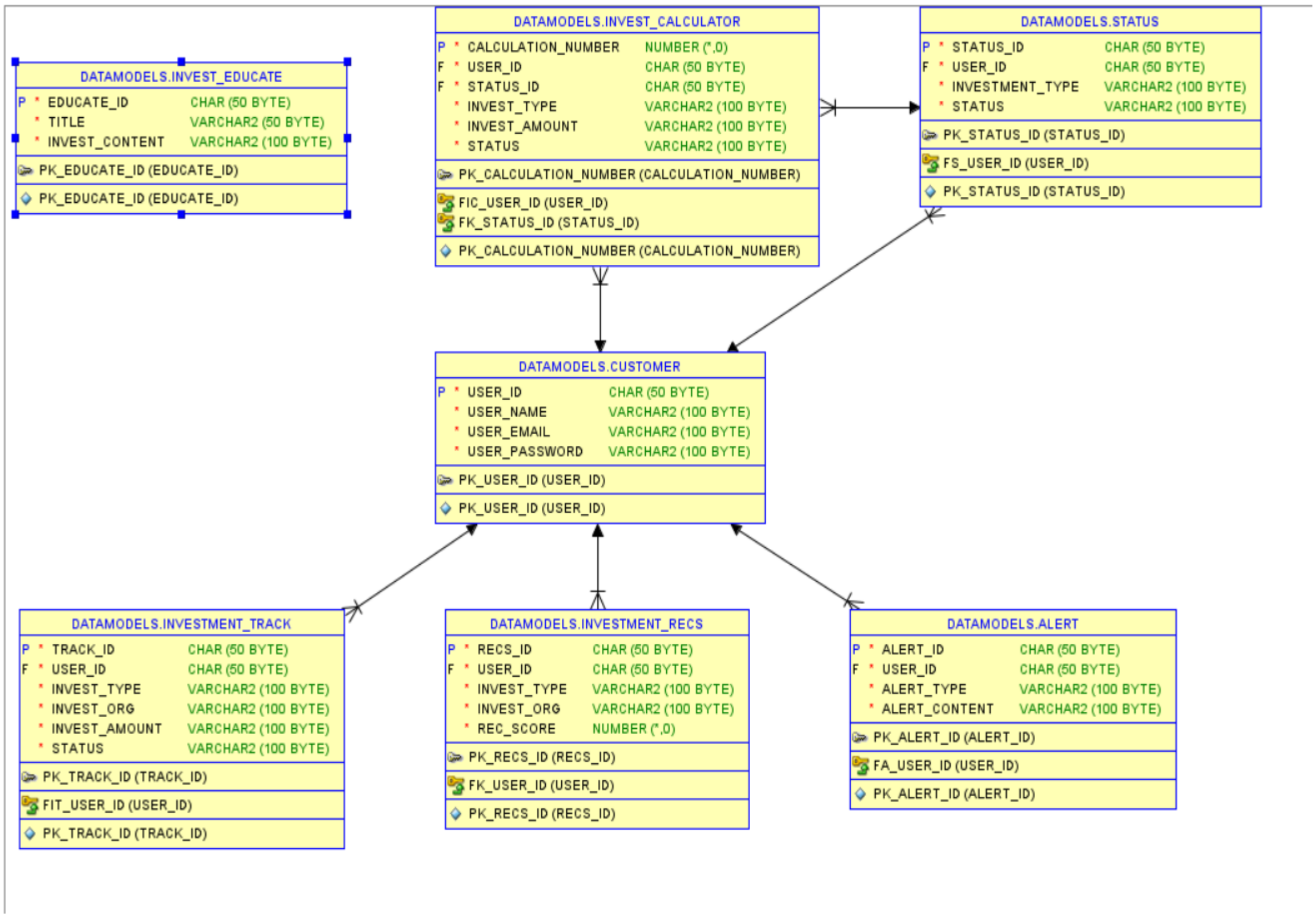
3.3 - SYSTEM PROCESS



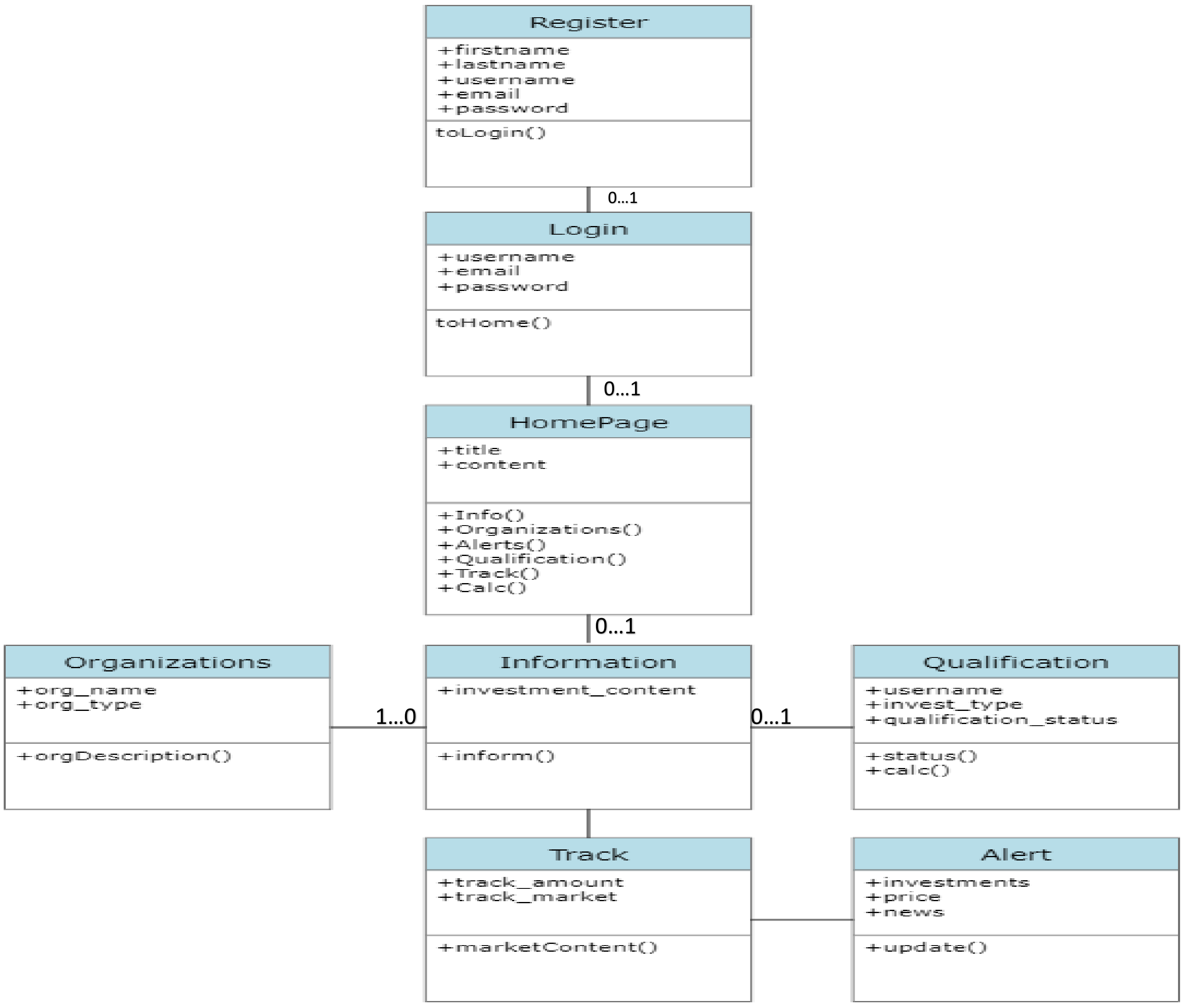
System Process version 2


Input
Validation process
Investment calculator process
Investment qualification check process
Investment recommendation process
Investment tracking and alert process
System maintenance and updates process
Output

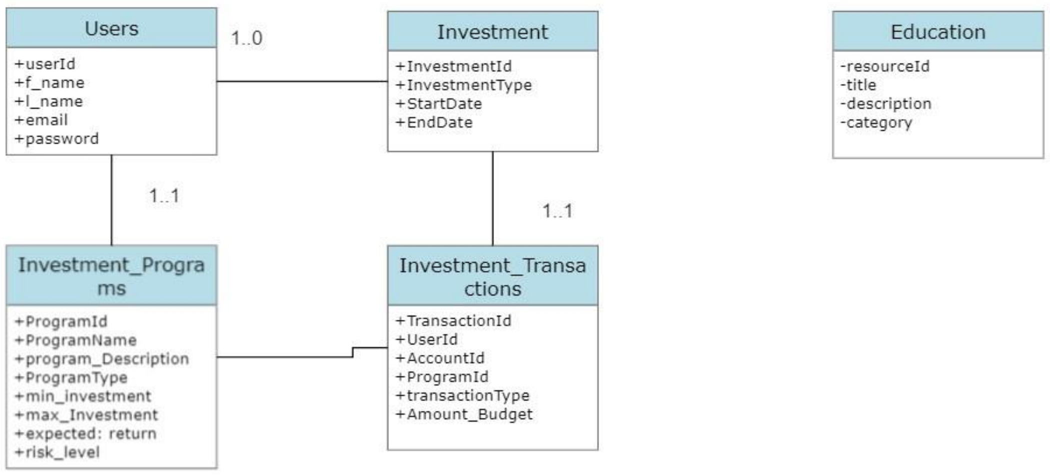
3.4 – Entity Relationship Tables


DATAMODELS.INVEST_EDUCATE
P * EDUCATE_ID CHAR (50 BYTE)
* TITLE VARCHAR2 (50 BYTE)
* INVEST_CONTENT VARCHAR2 (100 BYTE)
PK_EDUCATE_ID (EDUCATE_ID)
PK_EDUCATE_ID (EDUCATE_ID)
DATAMODELS.INVEST_CALCULATOR
P * CALCULATION_NUMBER NUMBER (*,0)
F * USER_ID CHAR (50 BYTE)
F * STATUS_ID CHAR (50 BYTE)
* INVEST_TYPE VARCHAR2 (100 BYTE)
* INVEST_AMOUNT VARCHAR2 (100 BYTE)
* STATUS VARCHAR2 (100 BYTE)
PK_CALCULATION_NUMBER (CALCULATION_NUMBER)
FIC_USER_ID (USER_ID)
FK_STATUS_ID (STATUS_ID)
PK_CALCULATION_NUMBER (CALCULATION_NUMBER)
DATAMODELS.STATUS
P * STATUS_ID CHAR (50 BYTE)
F * USER_ID CHAR (50 BYTE)
* INVESTMENT_TYPE VARCHAR2 (100 BYTE)
* STATUS VARCHAR2 (100 BYTE)
PK_STATUS_ID (STATUS_ID)
FS_USER_ID (USER_ID)
PK_STATUS_ID (STATUS_ID)
DATAMODELS.CUSTOMER
P * USER_ID CHAR (50 BYTE)
* USER_NAME VARCHAR2 (100 BYTE)
* USER_EMAIL VARCHAR2 (100 BYTE)
* USER_PASSWORD VARCHAR2 (100 BYTE)
PK_USER_ID (USER_ID)
PK_USER_ID (USER_ID)
DATAMODELS.INVESTMENT_TRACK
P * TRACK_ID CHAR (50 BYTE)
F * USER_ID CHAR (50 BYTE)
* INVEST_TYPE VARCHAR2 (100 BYTE)
* INVEST_ORG VARCHAR2 (100 BYTE)
* INVEST_AMOUNT VARCHAR2 (100 BYTE)
* STATUS VARCHAR2 (100 BYTE)
PK_TRACK_ID (TRACK_ID)
FIT_USER_ID (USER_ID)
PK_TRACK_ID (TRACK_ID)
DATAMODELS.INVESTMENT_RECS
P * RECS_ID CHAR (50 BYTE)
F * USER_ID CHAR (50 BYTE)
* INVEST_TYPE VARCHAR2 (100 BYTE)
* INVEST_ORG VARCHAR2 (100 BYTE)
* REC_SCORE NUMBER (*,0)
PK_RECS_ID (RECS_ID)
FK_USER_ID (USER_ID)
PK_RECS_ID (RECS_ID)
DATAMODELS.ALERT
P * ALERT_ID CHAR (50 BYTE)
F * USER_ID CHAR (50 BYTE)
* ALERT_TYPE VARCHAR2 (100 BYTE)
* ALERT_CONTENT VARCHAR2 (100 BYTE)
PK_ALERT_ID (ALERT_ID)
FA_USER_ID (USER_ID)
PK_ALERT_ID (ALERT_ID)

4 – Class Diagrams



Version 2


Users
+userId
+f_name
+l_name
+email
+password
1..0
Investment
+InvestmentId
+InvestmentType
+StartDate
+EndDate
Education
-resourceId
-title
-description
-category
1..1
1..1
Investment_Programs
+ProgramId
+ProgramName
+program_Description
+ProgramType
+min_investment
+max_Investment
+expected: return
+risk_level
Investment_Transactions
+TransactionId
+UserId
+AccountId
+ProgramId
+transactionType
+Amount_Budget