Bed & Breakfast

Reservation

and

Accounting System

Software Design Description

15 September 2015

Michael Straughan

Lead Software Engineer

CMIS 330 6381

Software Engineering Principles and Techniques

Ronald McFarland

**1. Introduction**

This section gives a scope description and overview of everything included in this Software Design Document. Also, the purpose for this document is described and a list of abbreviations and definitions is provided.

## **1.1 Purpose**

The purpose of this Software Design Document is to expand upon the Software Requirements Specification by defining an implementation of the "Bed and Breakfast Reservation & Accounting System". It should give all parties the necessary information of what is to be built and and how it is expected to be built.

## **1.2 Scope**

This scope is a summary from the previous SRS. The scope of the BBRAS software suite has two functions: Customer Reservations Processing and Managerial Reporting

The BBRAS system will enable customers to make an electronic reservation to stay at the facility. The system will also facilitate recording customer information, reservation information, payment information, customer account information, and payment/deposit verification.

The BBRAS system will also enable managers to pull reports from the accounting subsystem. Reports will be either a Reservation Report, which contains pertinent details about a specific reservation, and an Operational Report, which breaks down income, expenses and profit.

The goal of this software is to provide an electronic alternative to booking a reservation at this facility.

The scope is limited in that it does not provide any additional functionality, such as tracking of customers who are checked in, costs associated with business/facility operations, or as a digital homepage for the operation and will only handle reservation details and using those details provides reports to the management.

## **1.3 Definitions, Acronyms, and Abbreviations**

**SRS** – Software Requirements Specification

**SDD** – Software Design Document

**BBRAS** – Bed and Breakfast Reservation and Accounting System

**ERD** – Entity Relattionship Diagram

**ACD** – Architextural Context Diagram

**DFD –** Data Flow Diagram

**UI** – User Interface

**User** - Any person will interact with the software

**Customer** - The prospective user who will make a reservation within the software

**Manager/Management** - The user who will use the software to retrieve operational reports. This user might also be the contracting party for this software.

**Operational Reports** - A report produced by the software to display payment, expense and profit information. This information is pulled from the Accounting Database.

**Customer Database** - An external database that will record all customer information.

**Reservation Database** - An external database that will record all reservation information. This information is linked to the Customer Database entries.

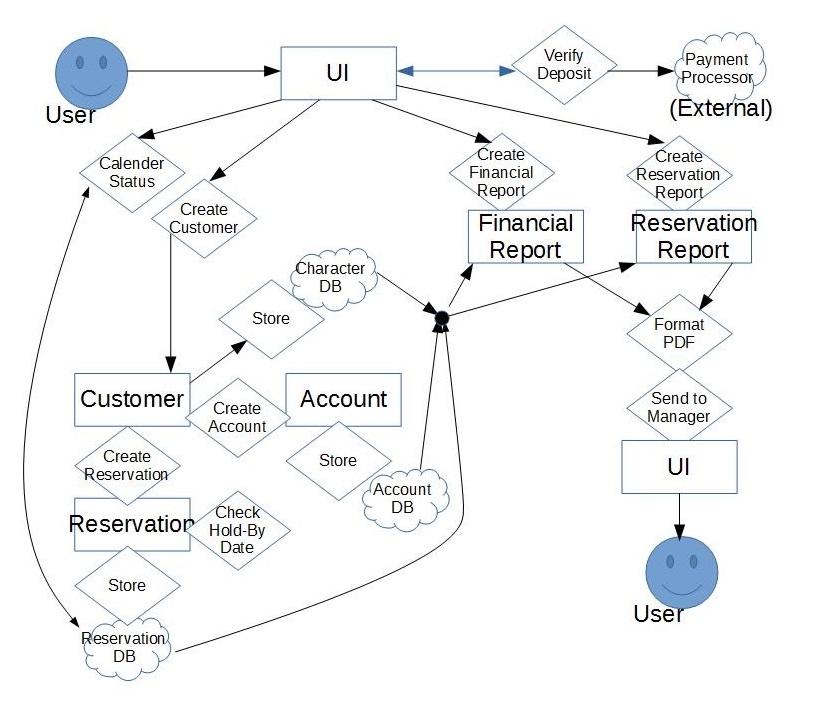
**Accounting Database** - An external database that will record all transactional data, such as payment for services. This information is linked to the Customer Database entries for tracking payment made/due. Also, it keeps a record of total payment, expense and profit information for the business/facility.

**Business/Facility** - The entity that utilizes this system.

## **2. References**

# **3. Decomposition Description**

The following figure is an Extended ERD. It is based off of the Class Diagram from the SRS document. This extended diagram is shows the relationship between the domain object classes, the relationships between each other through service objects and concurrent process objects.



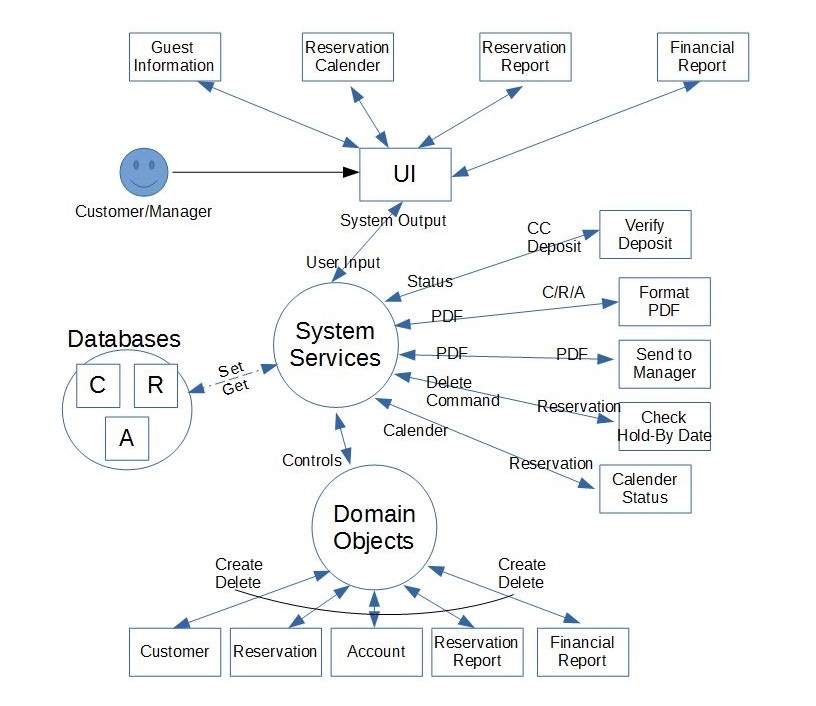
We then decompose the system using 3 categories: Module, Concurrent and Data.

## **3.1 Module Decomposition**

Using Modular Decomposition design, we split up the BBRAS into 3 modules: User Interface, System Services and Domain Objects.

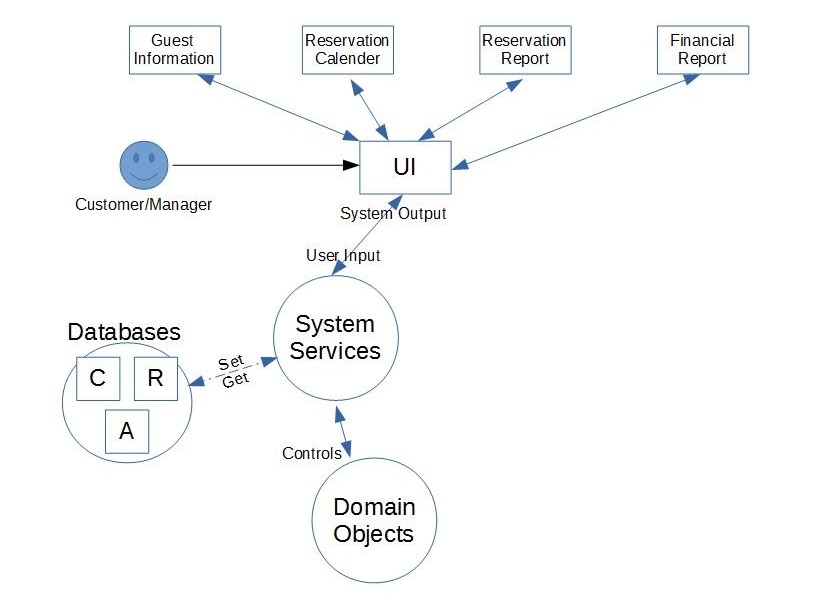
With this decomposition, we are able to segregate the different subsystems and further refine them within their category. By doing so, we will be able to define subsystem interfaces that will allow us to formalize how each subsystem talks to one another, and if needed, make corrections/updates/maintenance on a subsystem without affecting the rest of the system as a whole.

The following is an updated ACD that defines each Module Decomposition subsystem.



### **3.1.1 User Interface**

The User Interface module is how othe users of the system interact with the BBRAS system. The Users are the Customers and Management. The UI subsystem components are designed to be as limiting as possible in order to streamline user interaction with the system, yet comprehensive enough that everything needed for the user is available.



**3.1.1.1 Reservation Calendar**

This UI element will show a monthly calendar to the user, with each date showing vacancies. This should be the most universal way to show all availability with one UI element update rather than checking for individual dates one at a time.

**3.1.1.2 Guest Information**

This will show a guest information submission page to the user. This page will also collect payment information and the option to check if the user would like to make a deposit to guarantee the reservation slot.

**3.1.1.3 Reservation Report**

This will show a reservation report in PDF form of a specific reservation slot. This is only available to the management.

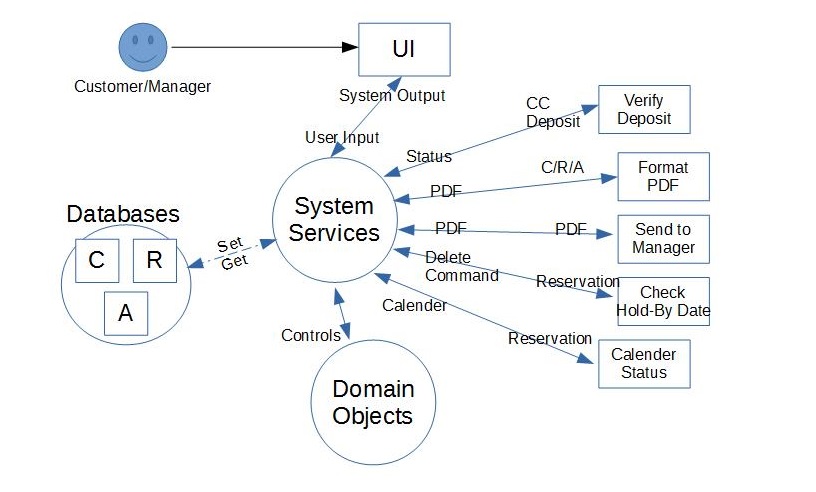
**3.1.1.4 Financial Report**

This will show a financial report in PDF form. This financial report will show the income, expenses and profit for the business.

\*\*\* Note: these concise UI elements will allow us to reduce the System Services described (not defined) in the previous SRS STD, eliminating the branch where a reservation will fail if the requested dates are not available. The Reservation Calendar should restrict the ability to select unavailable dates, making the system simpler and the user reservation process less frustrating.

### **3.1.2 System Services**

The System Services module houses all the processes to interact with the different subsystems, the User Interface module, the Domain Object module and the databases. These processes are modularized away from the rest of the system so that the BBRAS system logic is encapsulated into one single system. This should make development, maintenance and troubleshooting easier.



**3.1.2.1 Calender Status**

This process is used to get the reservation information from the databases, format it into an input for the UI to show the user what reservation dates/slots are available.

**3.1.2.2 Verify Deposit**

This process is used to verify the credit card deposit to guarantee a reservation slot. This process will ultimately communicate with an external payment processor. Depending on the verification outcome, a result will be pushed to the user through the UI. Refer to the SRS User Cases for specific results.

**3.1.2.3. Format PDF**

This process will take data from the databases and format them into human-readable PDF form. This will be called from the UI elements Reservation Report and Financial Report.

**3.1.2.4 Send to Manager**

This process will take the PDF from the Format PDF process and send it to the management, to the UI elements Reservation Report or Financial Report. This process is tightly coupled together with the Format PDF process.

**3.1.2.5 Check Hold-By Date**

This is a concurrent process that will be further defined in section 3.2

**3.1.2.6 Databases**

This is a subsystem of databases which hold all of Domain Objects module data objects. This is listed under the Services System because of the tight coupling between the processes and the data elements.

### **3.1.3 Domain Objects**

This is the Domain Objects subsystem. It contains class objects that hold it's relevant data and these objects are stored within the System Services databases.

See section 3.3 for specific details.

## **3.2 Concurrent Process Decomposition**

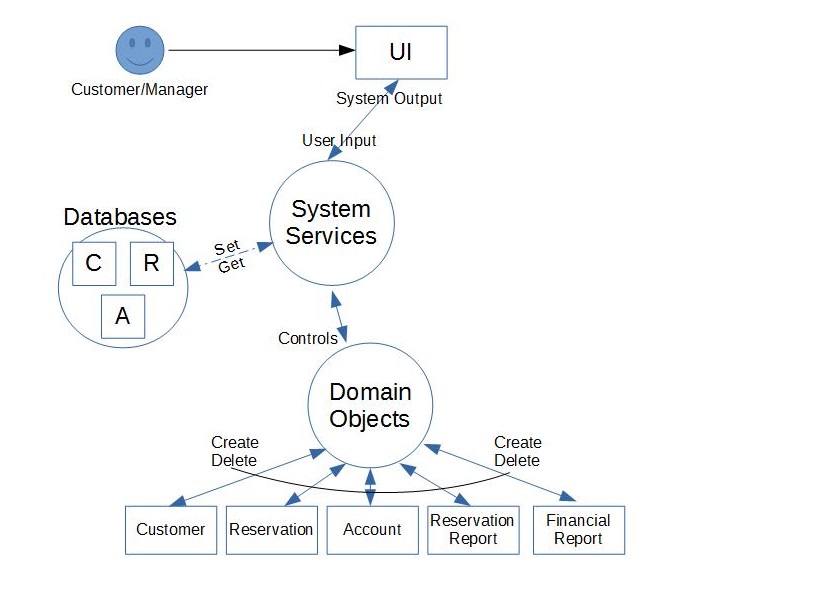
Concurrent Processes are processes that are user independent and run as a function of the system itself.

### **3.2.1 Check Hold-By Date**

This concurrent processes checks day by day when to drop the guaranteed status of a reservation. This is tied into whether a reservation has a deposit made to guarantee it or not. During the check, if HAS-DEPOSIT is FALSE and CURRENT-DATE == HOLD-DATE, then the reservation is dropped.

## **3.3 Data Decomposition**

The data decomposition defines the Domain Objects. Domain Objects contain data information relating to their module. This subsystem is segregated away from the UI and System Services to define data classes and their own functions. Again, encapsulation should make development/maintenance/debugging easier and more efficient.



### **3.3.1 Customer**

This class holds customer information: First Name, Last Name, Email, Phone No, Customer No, Reservation No, Account No. This class has a corresponding Reservation class and Account class tied to it.

**3.3.2 Reservation**

This class holds reservation information: Reservation No, Start Date, End Date, Room No, Deposit Status, and Hold By Date. This class is linked to a customer through a Customer Number.

**3.3.3. Account**

This class holds account information: Account No, Payment Made, Payment Due, Credit Card No. This class is linked to a customer through a Customer Number.

**3.3.3.4 Reservation Report**

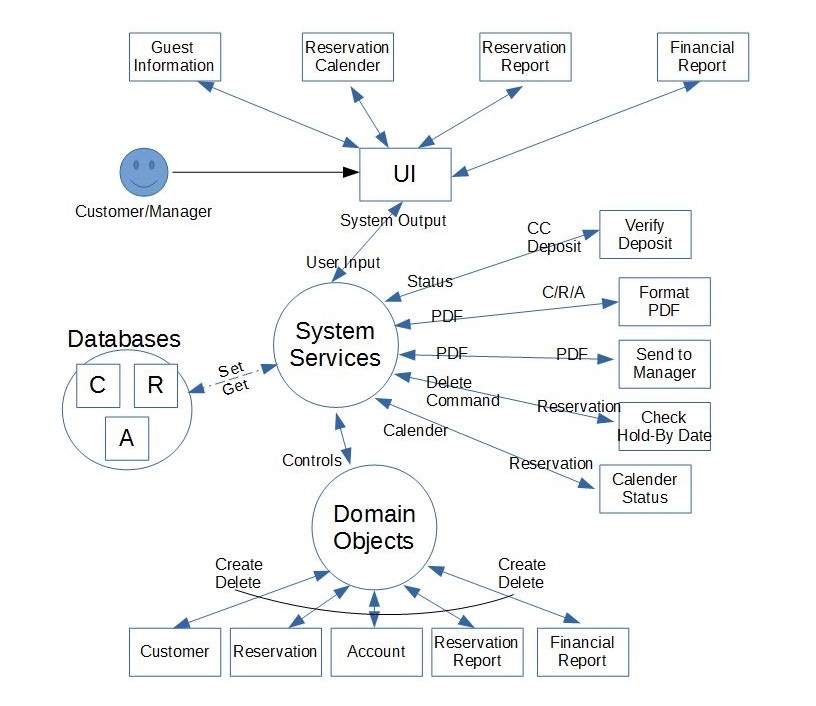
This class holds reservation information: Customer No, Reservation No, Account No. This report, through the various class identifier numbers, can create a summary of the customer, reservation and account for review by the management.

**3.3.3.5 Financial Report**

This class holds financial information: income, expenses, profit. This report can create a summary of the operations financial data for review by the management.

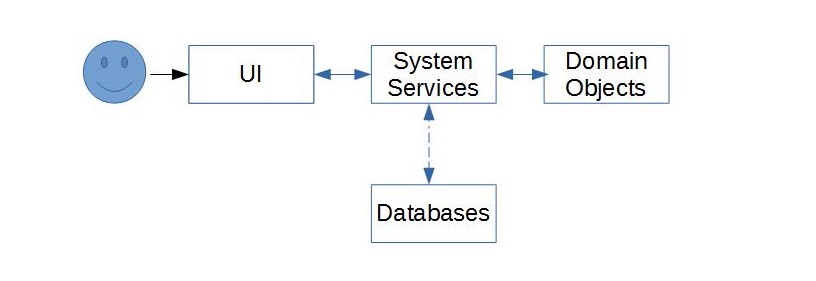
# **4. Dependency Description**

The subsystem dependencies is shown from this updated level 1 DFD below. The UI and the Domain Objects has dependencies with the System Services, and the System Services has dependencies with the UI, Domain Objects and Database subsystems. The Module, Process and Data dependencies are outlined below.



# **4.1 Inter-module Dependencies**

BBRAS is split into 3 modules, and the inter-module dependencies are shown in the diagram below.



### **4.1.1 User Interface Dependencies**

The UI takes user input and interfaces with the System Services module. The user solely relies on the UI to interact with BBRAS. The UI depends on the System Services and its processes to both display information and prompts to the user and to handle inputs from the user to a usable form for the system.

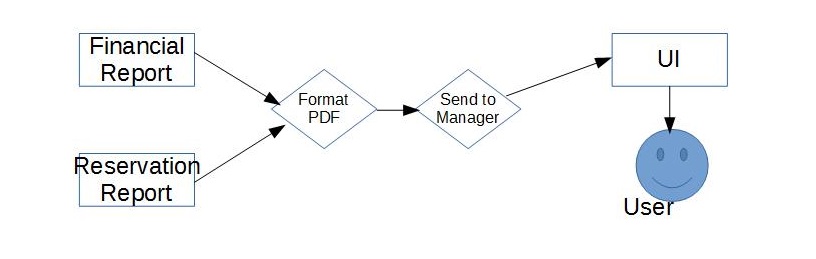
### **4.1.2 System Services Dependencies**

The System Services module processes input from the UI, creates the appropriate data objects through the Domain Objects module and stores/retrieves those data objects with the Databases subsystem. The System Services module is at the core of BBRAS and is dependent on all other modules.

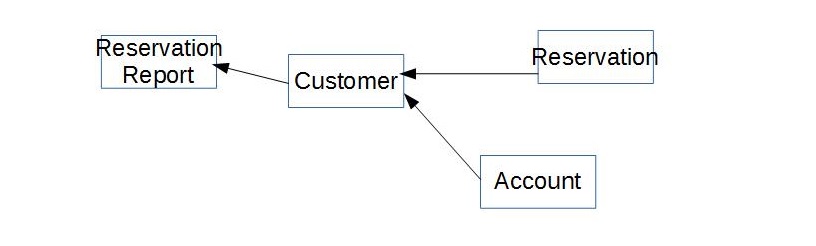
### **4.1.3 Domain Object Dependencies**

### The Domain Object module is controlled by the System Services module. Requests to it are made to create, delete and update data objects. Those data objects are used to create Reports, and store those objects in the Databases for record in use for the reservation and financial requirements.

## **4.2 Inter-process Dependencies**

The process dependencies are shown in the diagram above. The only process that has a dependency to another is “Send to Manager”. It requires that the “Format PDF” process has properly taken data from either the “Financial Report” or “Reservation Report” Domain Object and output to it a PDF. It will then take that PDF and output it to the UI module to display to the user.

**4.3 Data Dependencies**

From the diagram above, we can see that there are two data class objects that have dependencies to other data classes: Reservation Report and Customer.

### **4.3.1 Customer**

The Customer data class has dependencies to the Reservation and Account data classes. The reason for this dependency structure is that there for every Customer there needs to be a corresponding Reservation to keep track of the reservation dates and a corresponding Account to keep track of payment information. BBRAS will not process a user without all 3 data classes. The UI is responsible for ensuring that all information is available from the user. Also, there can only be 1 Reservation and Account for every Customer. A limitation of this dependency structure is that there cannot be more than one Reservation for each Customer. This is a system limitation and will not affect the end user; a user will still be able to make more than one reservation, the system will simply create multiple Customer data class objects for them.

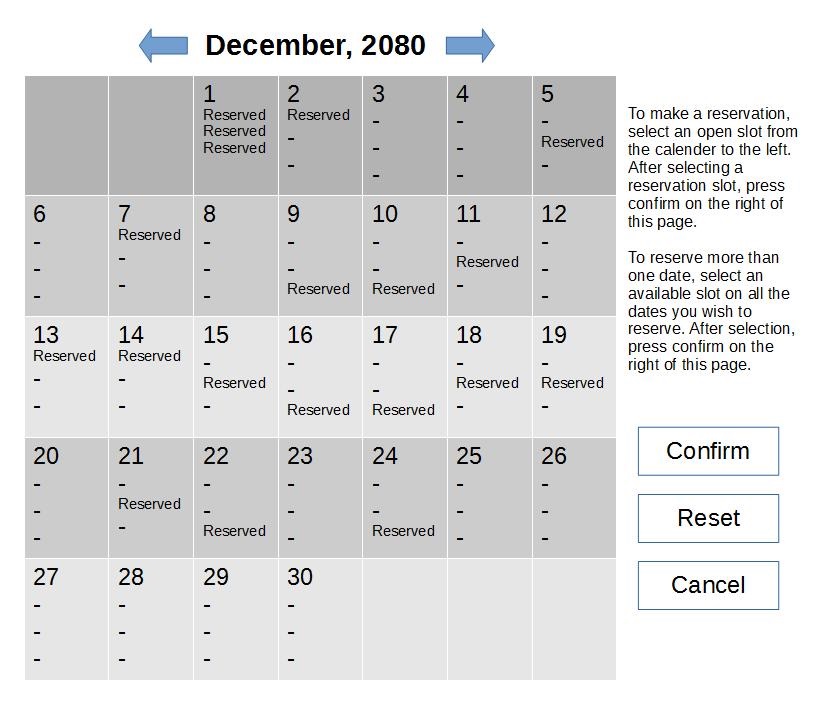
### **4.2.2 Reservation Report**

# The Reservation Report data class has dependencies to Customer, Reservation and Account data classes. This is required because in order to provide a full detailed report of a reservation, the customer and account information is required in addition to the reservation details.

# **5. User Interface Description**

The user interface is the way the user will interact with the system. The two different categories of users are Customers and Managers. A different UI will be displayed depending on the user of the system.

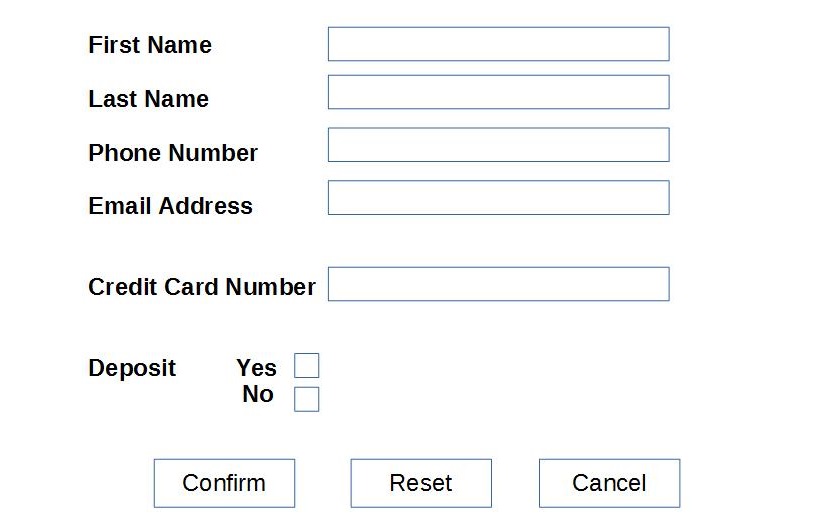
# **5.1 Reservation Calender**

****

### The Reservation Calender UI has 2 functions. First, the calender is used as a display to show the user which dates and which rooms are available for reservation. It is clear on any given day which rooms are vacant and which rooms are reserved. Second, the calender is used as a selection tool for the user to select the date or range of dates they wish to reserve.

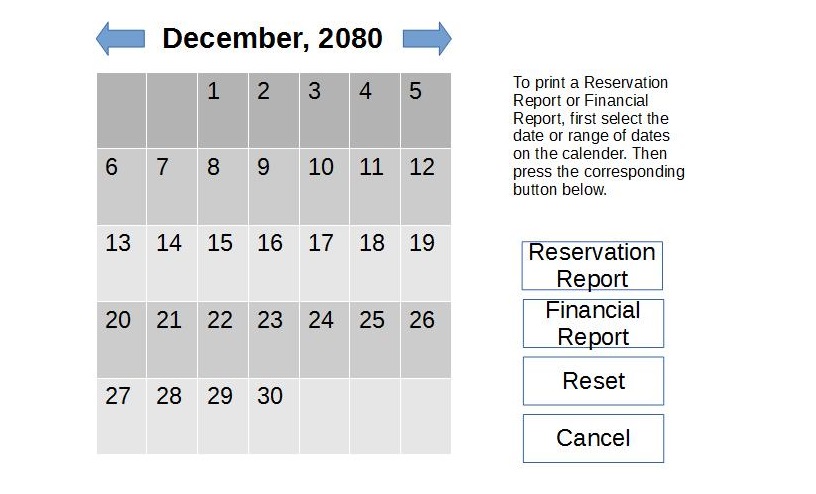
Instructions on how to use this UI is to the right of the calender and Confirm, Reset and Cancel buttons are below the instructions. These buttons names are indicative of their function.

### **5.2 Guest Information**

After a successful completion of the Reservation Calender UI, the user is presented with the Guest Information UI. The user is able to input all the required information into their resepctive text fields.

It is important to note that there are Deposit Yes-No check boxes for the user to select. If the user selects Deposit Yes, then upon pressing the Confirm button the UI will call the System Services “Verify Deposit” process and upon successful confirmation of a deposit, the BBRAS will continue with processing the reservation. If the verification of the deposit fails, then the user will remain on this page and a prompt will inform the user that the deposit was unsuccessful. These cases are outlined in the SRS.

### **5.3 Reservation Report & Financial Report**

The Reservation Report and Financial Report UI are actually one interface page that the manager user will use. A different UI is not created for each one because the same interface elements are used for both with the change of one button.

The Reservation Report and Financial Report UI enables the manager user to select a date or range of dates on the calender in the center of the UI and click on the report type they wish to receive. Instructions on how to use this UI is written out to the right of the calender and the buttons that execute functions are located below the instructions on the right side of the UI. The button functions are self explanatory with their descriptive names.

# **6. Detailed Design**

# **6.1 Module Detailed Design**

# **6.1.1 Verify Deposit**

This process is used to verify a pending deposit from the UI. When the user selects to make a deposit to guarantee their reservation, this checks if that deposit will go through or not.

Methods:

1. checkDeposit() - This method will send the deposit information, the user name, credit card information and deposit amount, to an external credit card payment processor. The processor should return either a successful or unsuccessful charge.

2. returnStatus() - This method will return to the UI the status of the deposit.

# **6.1.2 Format PDF**

This process is called from the UI to retrieve either a Reservation Report or a Financial Report. This process will use the getters from each data class to populate its PDF.

Attributes:

1. pdf : pdf – This will contain all the data from the appropriate reservation form.

Methods:

1. retrieveReservationReport() - This method will call the getters from the Reservation data class.

2. retrieveFinancialReport() - This method will call the getters from the Financial Report data class.

3. formatPDF() - This method will use the data pulled from the appropriate data class and format them into a user readable PDF.

# **6.1.3 Send to Manager**

This method will send the PDF created from the Format PDF process and send it to the UI

Methods:

1. sendToUI() - This method will get the PDF from the Format PDF process and send it to the UI. Exactly method is exactly how it is described.

# **6.1.4 Check Hold-By Date**

This process is called concurrently, independent of the user. This method will review the current reservations of the day and check their hold status. If their hold status does not exist, which means there was no deposit paid to guarantee the reservation, the reservation is deleted for that day.

Methods:

1. checkReservations() - This method checks the reservations for the current date, evaluates them to see if they have a true hold status, and if not, calls the delete method within that reservation.

# **6.1.5 Calender Status**

This process is called when displaying the Reservation Calender to the user. On process call, it will create an instance of the current calender to be viewed and output it to the UI.

Attributes:

1. calender[] : array – This is an array of reservations that will populate the entire calender of a given month. It is a simple list of binary values, 0 meaning a vacancy and 1 being a reservation.

Methods:

1. retrieveReservations() - This method will cycle through all the reservations between a given date range (which will be exactly 1 month, from the 1st to the last day). This method will then build up the calender[] array.

2. sendCalender() - This method will send the calender array to the UI.

# **6.2 Data Detailed Design**

# **6.2.1 Customer**

The Customer data class represents the customer's information on record.

Attributes:

1. firstName : string – This attribute is the first name of the customer.

2. lastName : string – This attribute is the last name of the customer.

3. email : string – Thisi attribute is the email address of the customer. It might be used for contact purposes.

4. phoneNo : string – This attribute is the phone number of the customer. Its datatype is string because there might be extensions or other non-integer symbols (dashes or parentheses) typed in by the customer.

5. reservationNo : int – This attribute is the corresponding reservation identifier. This is used to create a link between this data class and its reservation data class.

6. accountNo : int – This attribute is the corresponding reservation identifier. This is used to create a link between this data class and its account data class.

Methods

# 1. makeReservationNo() - This is called by this class to create a reservation. By calling the constructor for the reservation class from this one we are able to link its dependency without having additional processes.

2. makeAccountNo() - This is called by this class to create an account. By calling the constructor for the account class from this one we are able to link its dependency without having additional processes.

3. getters()/setters() : For firstName, lastName, email, phoneNo, reservationNo, accountNo – These are the getter and setter methods for the attributes listed above. These getters/setters will be used to retrieve relevant data for use elsewhere in BBRAS.

# **6.2.2 Reservation**

The Reservation data class represents the customer's reservation at BBRAS.

Attributes:

1. reservationNo : int – This value is an identifier from which we be able to reference this data class from others. This Reservation data class is linked to a Customer by this number.

2. startDate : date – This value represents the starting date for the reservation.

3. endDate : date – This value represents the ending date for the reservation. If the reservation is only for one day then the startDate and endDate will be the same value.

4. roomNo : int – This is the room number the reservation is under. The user will not know what room they reserve as it is not indicated in the UI when they are selecting their reservation, but the system will automatically assign the reservation to an available room by determining which reservation icon they selected. The top icon is room 1, the bottom icon is room 3.

5. deposit : boolean – The deposit attribute is a key that lets the system know if there is a Hold status on the room. If the deposit boolean is false, there is no hold status, and the reservation is not guaranteed.

6. holdByDate : date – This value will always be the last day of the reservation if the deposit boolean is set to true and the customer did infact make a deposit to guarantee the reservation.

Methods

# 1. generateReservationNo() - This is the constructor method for this data class.

2. deleteThis() - This function deletes this reservation if the situation arises where the reservation is not guaranteed by a deposit and another user selects that date and provides a deposit for the reservation. This function will then delete itself to open up the slot for the guaranteed customer.

3. getters()/setters() : For startDate, endDate, roomNo, holdByDate – These are the getter and setter methods for the attributes listed above. These attributes will be utilized in other areas of BBRAS, namely with the Calender Status process when the UI requests to display the Reservation Calender.

# **6.2.3 Account**

The Account data class represents the customer's payment account with BBRAS.

Attributes:

1. accountNo : int - This value is an identifier from which we be able to reference this data class from others. This Account data class is linked to a Customer by this number.

2. paymentMade : decimal - This value is how much the account has paid up until now. This amount will include deposits (if made).

3. paymentDue : decimal - This value is how much the account owes. This amount will be adjusted if a deposit is made.

4. creditCardNo : int - This is a credit card number from the Customer for payment purposes.

Methods

# 1. generateAccountNo() - This is the constructor method for this data class.

2. getters()/setters() - For paymentMade, paymentDue – These are the getter and setter methods for the attributes listed above. Note that there is not a getter/setter for accountNo as that is made upon class creation and should not be adjusted.

# **6.2.4 Reservation Report**

The Reservation Report data class is a collection of information to display the details for a given reservation. The information is intended to be used as a reference to help accommodate guests as they use the facility.

Attributes:

1. customerNo : int – This attribute contains a customer number. With this customer number we can then find out all the information regarding that customer by calling that referenced data class.

2. reservatioNo : int – This attribute contains a reservation number. With this reservation number we can find out all the information regarding that reservation by calling that referenced data class.

3. accountNo : int – This attribute contains an account number. With this account number we can find out all the information regarding that account by calling that referenced data.

With these 3 identifiers we have all the information regarding a customer, their reservation and their payment account details. This will give us a comprehensive report for that particular reservation.

Methods

# 1. getAllInfo() - This method is used to retrieve the customerNo, reservationNo and accountNo. These attributes are necessary for us to build a comprehensive Reservation Report.

# **6.2.5 Financial Report**

The Financial Report data class is a collection of data to display the finances for the BBRAS for review by the management. The information displayed will give the managers numbers for expenses and income for a given date range.

Attributes:

1. expenses : decimal – This attribute contains the expenses value for the Financial Report. This attribute is calculated by the in-class method getExpenses().

2. profits : decimal – This attribute contains the profits value for the Financial Report. This attribute is calculated by the in-class method getProfits().

Methods

# 1. getExpenses() - This method will determine what the expenses are for the given date range of this Financial Report. The algorithm to determine the expenses are a constant value multiplied by the number of days in the date range of this report.

2. getIncome() - This method will determine what the income are for the given date range of this Financial Report. The method will calculate how many reservations have paid or will pay during the date range and return the sum of that number.