Customer Relationship Management Software Proposal

Tesseract Software Solutions

Proposal for a CRM Solution

Aperture CRM

Alano Peirce

4-28-2022

Version 1.0

CONTENTS

A. INTRODUCTION

In response to the American Video Game Company's interest in our consulting services, Tesseract Software Solutions would like to submit a detailed proposal introducing a thorough and robust Customer Relationship Management (CRM) solution to replace the current CRM system used by the American Video Game Company (AVGC). This software solution, dubbed "Aperture CRM", was custom-designed to include as many of AVGC's requirements as possible. Please let us know if any adjustments to the solution are desired; we will be more than happy to integrate additional or alternative features into our application where possible.

A.1. PURPOSE STATEMENT

This document will detail several key aspects relating to the CRM solution that we are proposing to AVGC. These aspects include:

- The specific requirements (provided by AVGC) that we included in our solution
- A discussion of various software development methodologies (and our recommendation regarding the most appropriate development methodology for this project)
- Visual representations depicting the design of our proposed CRM application
- The specific tests that we performed on our CRM application prototype, as well as the results of those tests
- A list of all the sources utilized during the creation of this document

A.2. OVERVIEW OF THE PROBLEM

Due to AVGC's rapidly expanding customer base, their current CRM systems will soon become inadequate for managing said customers. Since their customer base *continues* to increase — as indicated by statistical data, which demonstrates a 42% increase in sales over the last two years — AVGC needs a new CRM solution quickly before the already-excessive workload reaches critical levels and becomes entirely too much for their current CRM systems to handle.

In addition, AVGC's current CRM systems are disconnected from each other; this lack of centralization promotes disorganization and disarray. For example, a miscommunication could easily occur between two disconnected systems which results in the loss or compromise of data. Additionally, AVGC's current CRM system makes it difficult to locate the appropriate tool to use for performing a particular CRM function — since there are so many distinct tools, it can be hard to discern which tool handles what function.

A.3. GOALS AND OBJECTIVES

We aim to achieve the following goals and objectives with our proposed CRM solution:

- The solution must be able to handle a wide range of aspects to do with customer relationship management.
- The solution must be easy-to-navigate, user-friendly, and organized in an intuitive manner.
- The solution must include a robust and multi-layered security system.
- The solution must include the ability to grant different roles and permissions to different users. As an extension of this, the solution must also allow for certain features to be restricted by role and/or permission.
- The solution must be able to share data with other systems through means of integration with those systems.
- The solution must include features that allow easier accessibility for users with disabilities.

A.4. PREREQUISITES

The following prerequisites must be completed before we can begin to develop the proposed CRM solution (Aperture CRM):

Number	Prerequisite	Description	Completion Date
1	Infrastructure Survey	We must determine the specifications of AVGC's internal infrastructure, predominantly the infrastructure governing their current CRM systems. This is necessary because we would like our CRM solution to keep intact as much of the existing infrastructure as possible, as per AVGC's preferences.	4 weeks before commencement of project
2	CRM Data Collection	We must collect all data that is currently housed in or used in AVGC's current CRM systems. In addition, we must collect any other data that is related to CRM or is relevant to any features that we plan to implement in our CRM solution. We will house this data, or at least a substantial portion of it, within our proposed CRM solution.	2 weeks before commencement of project

A.5. SCOPE

The following items are within the scope of our proposed CRM solution:

- The ability to generate detailed summary reports
- The ability to both soft-delete and hard-delete data
- The ability to record certain actions of individual users for purposes of logging and auditing
- The categorization of business contacts by type
- The ability to handle 2000 users in all, and at least 500 users concurrently

At this time, the following items are NOT within the scope of our proposed CRM solution:

- Forecasting
- Contract creation, approval, signing, and termination
- Integration with AVGC's Active Directory server

A.6. ENVIRONMENT

Our proposed CRM solution is compatible with the following environments:

- **<u>Devices:</u>**
  - **Personal computers** (desktop computers, laptops, etc.)
  - **Mobile phones**
  - **Tablets**
- **<u>Web Browsers:</u>**
  - **Google Chrome** version 101.0 (for Windows 10, Android 11, and iOS 15)
  - **Google Chromium** version 101.0 (for Windows 10, Android 11, and iOS 15)
  - **Mozilla Firefox** version 99.0 (for Windows 10, Android 11) or version 99.1 (for iOS 15)
  - **Internet Explorer** version 11 (for Windows 10)
  - **Apple Safari** version 15.3 (for iOS 15)
- **<u>Database:</u>**
  - **MySQL Enterprise Edition** version 8.0

- <u>**Cloud Computing Subscription:**</u>
    - **Amazon Elastic Compute Cloud (EC2)** (allows users to rent virtual scalable servers)
    - **Amazon Simple Storage Service (S3)** (allows users to rent scalable cloud storage) (only to be used if needed)

B. REQUIREMENTS

The American Video Game Company has outlined an assortment of requirements for their Customer Relationship Management (CRM) software. These requirements include the following items, which will be expanded upon in the following subsections:

1) The ability to generate detailed summary reports
2) The ability to both soft-delete and hard-delete data
3) The ability to record certain actions of individual users for purposes of logging and auditing
4) The categorization of business contacts by type
5) The ability to handle 2000 users in all, and at least 500 users concurrently

B.1. FUNCTIONAL REQUIREMENTS

One of the functional requirements delineated by the American Video Game Company (AVGC) is **the ability to generate detailed summary reports** (using either predefined criteria or user-determined criteria). Our CRM software solution will implement this functionality as follows:

Upon navigating to the “Reports” page of the application, the user will be able to see several reports already loaded up onto the screen. These reports will include visual representations (such as bar graphs, line graphs, or pie charts) to assist with readability. If this is the first time that the user has ever logged into their account, these reports are predefined reports (which are the same for every new user); otherwise (if the user *has* logged in previously), then these reports are the reports that the user has previously chosen to appear on their “Reports” page (which may include a mixture of predefined reports as well as user-defined reports).

To create a new report, a user can simply click on the button that says “Create Report”. Upon clicking this button, a pop-up window appears; on this new window, a user can provide the details regarding the report they would like to generate. Once the user has finished and clicked the “Create” button on the pop-up window, the pop-up window closes and the new user-defined report is visible on the “Reports” page of the application.

Users may also delete or modify the criteria for any of the reports displayed on the “Reports” page; they can achieve this by selecting the relevant report and then clicking either the “Modify” button or the “Delete” button on the Reports page of the application. If the user clicks the “Modify” button, a pop-up window will appear on which the user can modify the criteria used to generate the report. Alternatively, if the user clicks the “Delete” button, a confirmation dialog box (which asks the user whether they are sure they want to delete the report) appears on the screen. If the user confirms that they do indeed want to delete the selected report, then the report is deleted.

The data for all reports will be calculated by the database itself; in other words, a tailored SQL query will automatically be generated by the software based on the chosen criteria for the report (whether that criteria is pre-defined or user-defined), and the database will return a result table that contains the results of that SQL query. The data contained in this result table is then displayed as a single report (on the "Reports" screen of the application) in an easily-readable format.

Another functional requirement delineated by AVGC is **the ability to both soft-delete and hard-delete data**. This is a useful feature, as it prevents the accidental or unwanted permanent deletion of data. A “soft-delete” causes a particular record or item to no longer be visible in the application (for the vast majority of users); however, the “deleted” record or item still exists in the database. On the contrary, a “hard-delete” causes a particular record or item to be deleted from the database (which, as a consequence, also causes the particular record or item to no longer be visible in the application for any user). The soft-delete and hard-delete features will be implemented as follows in our CRM solution:

For any type of data that a) allows user deletion and b) affects more than just the user’s own account settings, the “Delete” button will cause a soft-delete to occur for most users. The only users who can perform a hard-delete on this type of data are the administrator account and any other accounts that have been granted the appropriate privileges.

For example, the “Delete” button connected to each individual table that displays Contacts, Products, Orders, and Support Tickets will only perform a soft-delete for the vast majority of users. For the administrator (as well as any other users who possess elevated privileges), these “Delete” buttons only perform soft-deletes as well; however, these users are able to view a “Trash” page in their application, and this page is divided up into two sections: “Items you deleted” and “Items other users deleted”. All items that the user themselves deleted are listed under the former section, while items that any other user of the CRM system deleted are listed under the latter section. From this “Trash” page, the user with elevated privileges is able to permanently delete any item; that is, they are able to hard-delete any of the items from the “Trash” page. They are also able to restore (that is, “un-delete”) any item on the “Trash” page; this removes the item for the “Trash” page and restores it to view (for all users) in the appropriate location of the application.

The items listed under the “Items other users deleted” section of the “Trash” page will be the same for all users who have elevated privileges (in this case, this refers to users who have the ability to perform hard-deletes). Once an item has been in the “Trash” for 90 days, it will automatically be permanently deleted. This provides an ample window of time to reverse unwanted deletes while also preventing the excessive consumption of storage resources by unneeded items.

An additional functional requirement detailed by AVGC is **the ability to record certain actions of individual users (for purposes of logging and auditing)**. Our CRM solution will implement this functionality as follows:

All user login attempts (whether successful or unsuccessful) will be recorded in a log. Each record will also delineate whether or not the login attempt was successful, the UTC timestamp at which the login attempt occurred, and the username utilized for the login attempt. Logging these activities and items will be very useful for future auditing purposes (regarding both proactive and reactive auditing). For example, the log can be used to prove that a certain user logged in when they said they did, or that they *didn't* log in when they were supposed to. Additionally, attempts at hacking into accounts using brute force can easily be caught by proactively monitoring the log for certain suspicious patterns (such as a succession of multiple unsuccessful login attempts using different, and perhaps nonexistent, usernames; or a succession of multiple unsuccessful login attempts using the same username).

In addition, all significant actions performed by each user — for example, adding, modifying, or deleting a record from the Contacts, Products, Orders, or Support Tickets tables — will also be recorded in a log. Less significant actions (such as adding or deleting reports on a user's own Reports page) will not be recorded, since these actions do not have substantial or large-scale implications. Recording these actions will also assist greatly with future auditing or security concerns. For example, if a number of Contact records are created that contain only gibberish, a quick look at the log can easily identify the user responsible for creating those records. Similarly, if unwanted deletions occur, the user responsible for said deletions can be promptly determined from the log.

A further functional requirement specified by AVGC is **the categorization of business contacts by type**. Our CRM solution will accomplish this by segmenting the business contacts into four categories — "Customers", "Suppliers", "Employees", and "Other". The "Customers", "Suppliers", and "Other" subsections will themselves be broken up into two separate categories — "Individuals" and "Businesses". In a more functional sense, our CRM software will implement these features as follows:

The "Main Menu" page of the application will include a "Contacts" button. If a user clicks on this button, they are brought to a different page of the application, on which they can click one of four buttons (each of which refers to a different type of Contact) — "Customers", "Suppliers", "Employees", or "Other". Upon clicking one of these buttons, the user is brought to a screen that displays a table populated with instances of the chosen type of contact. For the "Customers", "Suppliers", and "Other" tables, there are three display options (in the form of radio buttons positioned above the table) that the user can choose between — "All", "Individuals", or "Businesses". The "All" display option causes the table to display both individuals and businesses (all of which are AVGC contacts of the type that the user previously selected [i.e. they are all contacts of type "Customers", "Suppliers", or "Other", depending on what the user previously selected]). On the other hand, the "Individuals" display option causes the table to display only the individuals (persons) from that "All" list, while the "Businesses" display option causes the table to display only the businesses from that "All" list. Additionally, the user can add, modify, or delete a record for the table they are currently viewing.

Furthermore, any individual of a particular contact type can be associated with a business that is also of that contact type — for example, if Bob Smith is a customer of AVGC and he works for a business called "Game World", which is *also* a customer of AVGC, then it should be discerned whether or not Bob's customer status is for personal or business reasons; if the latter is the case, then it should be noted in Bob Smith's record (by way of database relationships) that Bob Smith works for Game World. That way, if AVGC needed to contact Game World, they could simply find the appropriate person(s) to contact by searching for the records of individuals who work at Game World (the implication being that these individuals are the appointed representatives for Game World).

B.2. NONFUNCTIONAL REQUIREMENTS

One of the nonfunctional requirements specified by AVGC is that **the software must have the ability to handle 2000 users total, and at least 500 users concurrently**. This means that the software must be able to handle these requirements without crashing or become excessively slow (we have chosen to define "excessively slow" as a response time greater than five seconds).

Our CRM solution implements these features by taking advantage of the scalability provided by a subscription to an Infrastructure-As-A-Service (IAAS) cloud computing service. When necessary (e.g. during peak times), additional web servers (provided by the cloud provider) are used to handle the additional traffic to the application. When not needed (i.e. during times when the traffic to the application is light), the cloud services (specifically, the usage of additional web servers) are not used.

AVGC currently possesses only one small web server, and this server is not sufficient to handle the levels of traffic described. Thus, a subscription to a minimal IAAS plan is the simplest and most cost-effective solution available. Since we have built the CRM software to be compatible with Amazon Web Services (AWS), which is a well-known cloud provider, a subscription to a minimal AWS plan — one that includes, at the very least, web servers — is required before AVGC can deploy our CRM solution.

C. SOFTWARE DEVELOPMENT METHODOLOGY

In the following subsections, we will discuss the advantages and disadvantages of two different software development techniques — 1) the waterfall method, and 2) the sashimi method.

C.1. ADVANTAGES OF THE WATERFALL METHOD

Because the waterfall method necessitates the completion of each development phase before moving on to the next development phase (Stephens 270), it is very intuitive and allows for a high level of clarity. These attributes provide a foundation for good organization, which is key to the efficient and successful completion of the project. When using the waterfall method, the developers on the team concretely know what phase of the project they are currently in — there is no confusion or ambiguity. This becomes even more crucial if a new developer (or group of developers) joins the software development team in the middle of the project (Gaille 2020).

In addition, the waterfall method allows for the easy enforcement of time constraints for the various development phases of the project (Gaille 2020). This is because it is much easier to keep track of the amount of time spent on a particular development phase when the development phases occur strictly consecutively and do not overlap with each other.

Furthermore, the waterfall method ensures that all requirements for the project are fully outlined and agreed upon by all parties before continuing with the project — in fact, this process is cemented as a development phase in the waterfall model, meaning that the development team *cannot* proceed with software development if all requirements have not been identified and agreed upon. This is important because it prevents requirements from being added or changed once software development is underway; such an occurrence could necessitate a massive overhaul and major changes to the system, essentially discarding hours of work that were already put into developing the software.

C.2. DISADVANTAGES OF THE WATERFALL METHOD

Because the waterfall method does not allow overlap of the various software development phases, developers working on the project cannot “work ahead”. This means that, even if everyone except one person on the development team has completed their contribution towards the current phase, *none* of the developers are able to move on to the consequent development phase until that one developer completes their contribution towards the current development phase. As a consequence, valuable time and manpower is wasted. Because of these shortcomings, the waterfall method often leads to lengthier development timeframes as compared to other software development methods (Gaille 2020).

Additionally, because the waterfall method only supports a forward progression through the software development phases (Stephens 271), it leaves no room for the reevaluation of previous development phases. Even the most meticulously-planned and carefully-built project can encounter unforeseen obstacles or issues that may warrant alterations relating to a previous development phase; however, if the waterfall method is being used, these alterations would not be possible, thus warranting an unnecessarily complicated alternative solution.

Furthermore, the waterfall method does not allow for testing (“verification”) until the software has been fully constructed. This can be rather perilous, as any adverse behavior uncovered during testing may warrant a massive overhaul and revision of the project (since so much of the project’s infrastructure — in fact, the entirety of it — has essentially been cemented in place already) (Gaille 2020). As a result, valuable time and resources can easily be wasted due to the waterfall method’s inefficient testing methodology.

C.3. ADVANTAGES OF THE SASHIMI METHOD

Because the sashimi method allows overlap of the software development phases, individual developers working on a project can “work ahead” (i.e. move on to the next phase) if they have completed their contribution towards the current phase. This allows for an efficient and time-conserving system — for example, while some developers may still be finalizing the design for a certain part of the software, other developers can move on to programming the functionality (a.k.a. they can begin the “implementation” stage) for other parts of the software where the design has already been completed (Stephens 272).

Additionally, the sashimi method allows for regression through the software development phases (Stephens 273). This is extremely useful for a multitude of reasons. For example, it is entirely possible that the requirements for the project may need to be altered due to an unsurmountable obstacle encountered during the “Design” or “Implementation” phase of development. Similarly, it may be discovered during the “Implementation” phase that there is a more intuitive or convenient layout for the design of the software (which relates to the “Design” development phase). In both of these scenarios, the sashimi method allows for necessary changes that relate to previous development phases.

Furthermore, the sashimi method allows for testing (“verification”) of the software *while* it is being constructed. This allows issues to be quickly identified and resolved after the implementation of particular features, which prevents massive overhauls of the software that can become necessary if an issue occurs but is not detected and rectified quickly.

C.4. DISADVANTAGES OF THE SASHIMI METHOD

Since the sashimi method allows overlap of the various software development phases, a project can quickly become disorganized. Many different developers could be working on many different development phases of the project, and it becomes easy to lose track of what each developer is working on. This can become especially problematic if new developers join the team midway through project development — being met with such disorganized chaos is likely to be overwhelming, disorienting, and/or demoralizing to these new developers.

Additionally, because the sashimi method does not require a sequential and isolated walkthrough of the software development phases, it becomes more difficult to enforce time constraints on the various development phases of the project. Since phases don't have to be completed all at once and can be revisited at a later point in time, it can be challenging to keep track of how much time is spent on a particular development phase. This can lead to poor time management of the project development phases, an unfortunate consequence resulting from the inability to accurately gauge productivity levels.

Furthermore, because the sashimi method allows the development team to begin designing and building the project even before the requirements for said project have been finalized (Stephens 272), it can lead to situations where — if the requirements are modified after the designing and/or building process of the project have already begun — large portions of the project may have to be revised or even rebuilt from scratch. This results in a pointless drain on the development team's time and effort.

C.5. BEST SUITED

Since AVGC is beginning to outgrow their CRM systems, those systems will soon become inadequate for managing their quickly-expanding customer base (which has increased 42% over the course of the last two years). In order to maintain satisfactory levels of customer service and business organization, it is crucial that AVGC deploys a new CRM system before their customer base grows entirely too large for their current CRM system to handle.

Due to this time constraint, the waterfall method may not be the best software development technique for this project. Owing to its previously-discussed disadvantages, the waterfall method often leads to a longer development process than other methods; this means that AVGC may need to wait longer than necessary for the delivery of their new CRM system. The sashimi model, on the other hand, is essentially a fast-tracked version of the waterfall method — it allows for development phases to overlap, which conserves time and resources.

Because of these reasons, we recommend that AVGC reconsider their decision to use the waterfall method for the development of this project; instead, we recommend the sashimi model, which will likely produce a finished product in a quicker timeframe than the waterfall model.

D. DESIGN

The following subsections provide graphics that go into detail regarding specific aspects of Aperture CRM.

D.1. STORYBOARD DEPICTING SCREENS ACCESSIBLE FROM APERTURE CRM'S MAIN MENU

The following graphic displays a storyboard that details all of the application screens accessible from Aperture CRM's "Main Menu" screen. Each arrow on the storyboard extends from a particular button on the "Main Menu" screen; the arrow indicates what application screen the user is redirected to upon clicking that button.

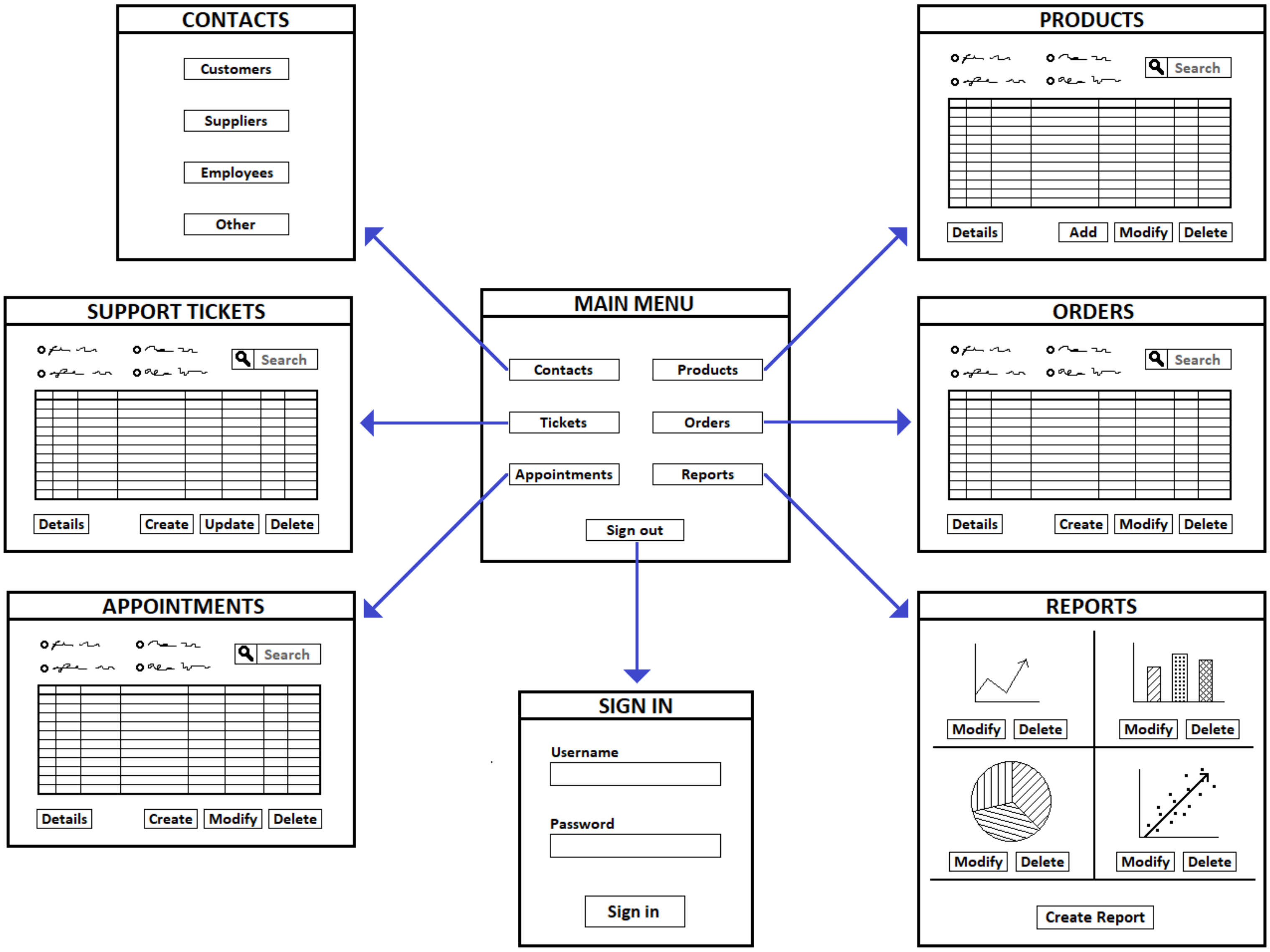

Figure 1: Storyboard displaying the screens accessible from Aperture CRM's "Main Menu"

D.2. GUI OF APERTURE CRM'S "MAIN MENU" SCREEN

The following graphic displays the finalized GUI (graphical user interface) of Aperture CRM's "Main Menu" screen.

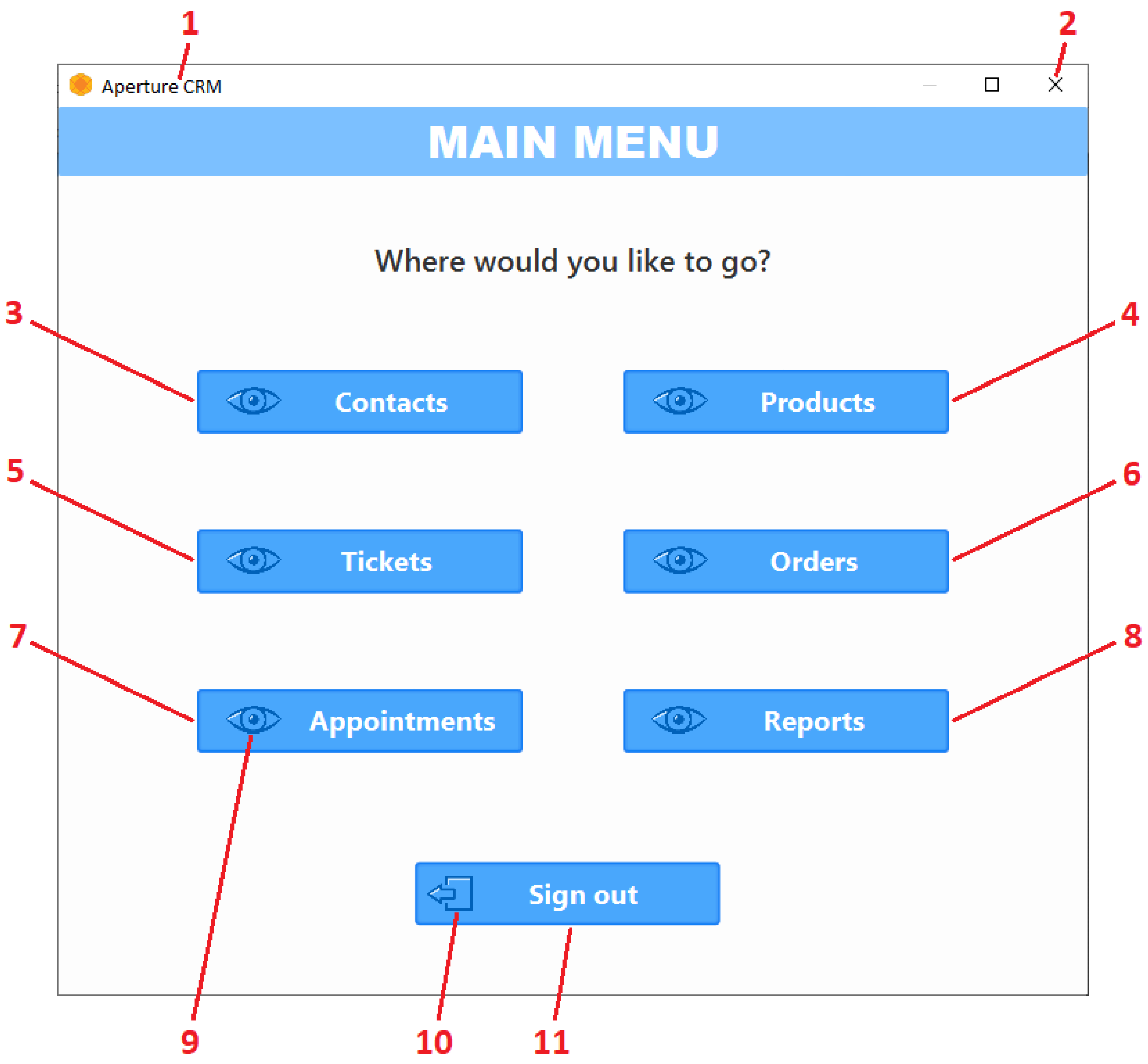

Figure 2: GUI of Aperture CRM's "Main Menu" screen

GUI Control Mapping			
ID	Control	Property	Data Source
1	Application Title	Text = “Aperture CRM”	Internal Variable
2	Exit Button	Upon clicking this exit button, the application closes.	Internal Method
3	Button	Upon clicking this button, the user is navigated to a screen where they can choose between four types of business contacts — “Customers”, “Suppliers”, “Employees”, and “Other”.	Internal Variable
4	Button	Upon clicking this button, the user is navigated to the application’s “Products” screen, which details all of AVGC’s current product offerings (and allows the user to add, modify, and delete products).	Internal Variable
5	Button	Upon clicking this button, the user is navigated to the application’s “Support Tickets” screen, which details all of AVGC’s support tickets (both resolved and unresolved) from the last 90 days. The “Support Tickets” screen also allows the user to create, update, and delete support tickets.	Internal Variable
6	Button	Upon clicking this button, the user is navigated to the application’s “Orders” screen, which details all of AVGC’s unfulfilled orders as well as all fulfilled orders from the past 90 days. The “Orders” screen also allows the user to create, modify, and delete orders.	Internal Variable
7	Button	Upon clicking this button, the user is navigated to the application’s “Appointments” screen, which lists all of AVGC’s future appointments as well as any past appointments that occurred during the past 90 days. The “Appointments” screen also allows the user to create, modify, and delete appointments.	Internal Variable
8	Button	Upon clicking this button, the user is navigated to the application’s “Reports” screen, which displays the reports that the user has previously chosen to appear on their “Reports” screen (this can include both the default predetermined reports as well as user-defined reports). The “Reports” screen also allows the user to create new reports as well as modify or delete existing reports.	Internal Variable
9	ImageView	Image = 👁; it is used to signify that the associated button will allow the user to <u>view</u> the specified items (e.g. in this case, they will be viewing appointments and the relevant details of those appointments).	The image file will be hosted on AVGC’s web server, but it will not be publicly accessible
10	ImageView	Image = ⇦ ; it is used to signify that the associated button will cause the user to <u>leave</u> (i.e. sign out of) the application.	The image file will be hosted on AVGC’s web server, but it will not be publicly accessible
11	Button	Upon clicking this button, the user is signed out of the application (they are redirected back to the Login Screen, and must sign in again if they wish to continue using the application).	Internal Variable

E. TESTING

The following subsections detail different tests that we ran on the Aperture CRM application. The results of those tests are also detailed.

E.1. UNIT TEST

A **unit test** is a test that checks the functionality of a particular piece of code (Stephens 179). The following tests are **unit tests** that we performed on our application.

E.1.1. TEST TO VERIFY APPLICATION'S ABILITY TO PROPERLY SWITCH SCREENS

Requirement to be tested: In this test, we will be evaluating our application's ability to properly switch screens. We will be testing ALL places in the application where a screen switch occurs.
Preconditions: • The GUI (graphical user interface) of all screens in the application must already be fully implemented. • The ability of the application to switch screens must already be programmed into the application in all applicable places.
Steps: **1)** First, the tester must make a flowchart or storyboard depicting ALL places in the application where a screen switch occurs. This flowchart/storyboard must indicate the event that triggers each screen switch, as well as the screen that should be switched to (for each instance of a screen switch). This is to ensure that no mistakes are made during runtime. **2)** Then, the tester must load up the application and log in using either the test or administrator account. **3)** After logging in, the application should have switched from the Login Screen to the Main Menu. The tester should record whether or not this occurred, as well as any relevant details. **4)** Starting from the Main Menu, the tester must now perform each triggering event on every screen. The easiest way to ensure that ALL triggering events in the entire application are performed is to follow each branch from the Main Menu all the way to the end before starting on a new branch from the Main Menu (the tester should reference the previously-

created flowchart or storyboard, which should be of great assistance for this task). For each triggering event performed, the tester should record the following items:

- **a.** Whether or not the application switched screens
- **b.** If the application did switch screens, record what screen it switched to.
- **c.** If the application did switch screens, record whether or not the application switched to the *appropriate* screen.

Expected results:

Upon the occurrence of a triggering event (e.g. clicking a certain button or pressing the "Enter" key), the application should switch to the proper screen (which will differ depending on what the triggering event was and what screen the application was previously displaying).

Pass/Fail:

Our application **passed** this test. All screen switches in the application occurred properly.

E.1.2. "ADD CUSTOMER" TEST

Requirement to be tested:

This test will verify that users are able to create a new customer record using our application.

Preconditions:

- The "Customers" table (a.k.a. the table on the application's "Customers" screen where users can view AVGC's customer records) must already be fully implemented.
- The "Add Customers" screen (where customers input customer information for the new customer record they would like to create) must already be fully implemented.
- All buttons along the path from the "Main Menu" screen to the "Add Customers" screen must work properly. This includes the ability to access the "Main Menu" screen, the proper functioning of the button used to access the "Contacts" screen (the "Contacts" button on the "Main Menu" screen), the proper functioning of the button used to access the "Customers" screen (the "Customers" button on the "Contacts" screen), and the proper functioning of the button used to access the "Add Customers" screen (the "Add" button on the "Customers" screen).
- The ability for users to create new customer records must already be fully implemented.

Steps:

1) First, the tester must navigate to the "Add Customer" screen of the application. The proper sequence of buttons to click in order to achieve this (starting from the Main Menu) is Contacts → Customers → Add.
2) Then, the tester must populate the text fields on the "Add Customer" screen with appropriate input. The input in each text field will become a particular detail stored in the new customer record.
3) Next, the tester must click the "Add Customer" button positioned at the bottom of the "Add Customer" screen.
4) Afterwards, the tester must check the table on the "Customers" screen to determine whether or not they can see the new customer record that they just created. If so, the tester must determine whether or not the details of that customer record match up with the details that the tester provided earlier.
5) Finally, the tester must check the database to see whether or not the new customer record has been added to the database. If so, the tester must check that the details of that customer record match up with the details that the tester provided earlier.

Expected results:

The tester should be able to successfully create a new customer record. This new customer record should be automatically saved to the database, and it should also be visible in the table located on the "Customers" screen of the application. Furthermore, the details of the new customer record must match up with the details that the tester provided.

Pass/Fail:

Our application **passed** this test. The tester was successfully able to create a new customer record, which was automatically saved to the database and was also visible in the table located on the "Customers" screen of the application. Additionally, the new customer record accurately reflected the details provided by the tester.

E.2. AUTOMATED TEST

An **automated test** is a test that is performed automatically (often by an automated testing tool). The test case itself is predetermined (by a person), but the automated testing tool can run that test over and over again using differing criteria (Stephens 182).

In the following subsection, we will be using automated testing to run a **load test**, which is a type of test that measures an application's ability to handle a certain number of users (Stephens 182).

E.2.1. LOAD TEST

Requirement to be tested:

With this test, we will be evaluating our application's ability to handle 500 concurrent users.

Preconditions:

- The application must be fully developed (in both a design sense and a functionality sense) before this test can be run.
- A script must be written that can simulate users logging in and performing functions in the application. This can be done using automated testing software.

Steps:

1) First, the script must be run to simulate 100 users logging in and performing various tasks in the application.
2) Then, the automated testing software must keep track of the response times experienced by each user; if a response time for any user surpasses five seconds, the automated testing software should make a note of this. The maximum response time experienced by any user should also be recorded by the automated testing software.
3) Next, the script should be run again to simulate 300 users logging in and performing various tasks in the application.
4) Again, the automated testing software must keep track of the response times experienced by each user; if a response time for any user surpasses five seconds, the automated testing software should make a note of this. The maximum response time experienced by any user should also be recorded by the automated testing software.
5) After that, the script should be run again to simulate 500 users logging in and performing various tasks in the application.
6) Finally, the automated testing software must once again keep track of the response times experienced by each user; if a response time for any user surpasses five seconds, the automated testing software should make a note of this. The maximum response time experienced by any user should also be recorded by the automated testing software.

Expected results:

If this test is successful, the application will be able to handle 500 distinct users simultaneously performing tasks without any part of the application (including the server(s) and database) crashing or otherwise malfunctioning. In addition, the application's response time for each user must not, any point, exceed five seconds.

Pass/Fail:

Our application **passed** this test. The application was able to handle 500 simultaneous users without crashing or malfunctioning in any way. Additionally, the slowest response time recorded was 3.92 seconds.

F. SOURCES

- Gaille, Louise. “15 Advantages and Disadvantages of a Waterfall Model.” *Vittana.org*, 19 Mar. 2020, https://vittana.org/15-advantages-and-disadvantages-of-a-waterfall-model.

- Stephens, Rod. *Beginning Software Engineering*, John Wiley & Sons, Incorporated, 2015. *ProQuest Ebook Central*, https://ebookcentral.proquest.com/lib/westerngovernors-ebooks/detail.action?docID=1895174.