Contents

[1.0 Introduction 2](#_Toc311566527)

[1.1 Purpose of this Document **Error! Bookmark not defined.**](#_Toc311566528)

[1.2 Scope of the Development Project **Error! Bookmark not defined.**](#_Toc311566529)

# Introduction

# Revised Requirements

# Data Design

# Architecture Design

# Detailed ~~Class~~ DFD Design

## Decision to Employ a Classless Design

There are many ways to conceptualize the solution to a programming problem, the most popular among them being an object-oriented approach employing a class-based design. However, after careful consideration we feel that although it may be possible to use an OO approach to achieve the goals of the ACT system, doing so would be become somewhat awkward and inelegant because of the nature of the project.

This project is very much web-based and rooted in the mechanics of PHP and HTML. We feel that it would be difficult to justify the existence of a class design based upon the lack of common properties and methods across the entities of the system. We do not want to force a class to generalize the behavior of disjoint entities in the system, which subsequently would only produce a class template with low cohesion.

It is true that there are some elements in the ACT system which would benefit from an OO approach (such as a Person class to represent the staff or customers). However, we feel that the system as a whole – particular with respect to how the entities interact and communicate – would be better analyzed using a different methodology. Therefore, rather than a class diagram we choose to represent our system via a Functional Modeling paradigm such as Data Flow Diagrams which clearly illustrate the actors, entities and how they flow and communicate within the context of a specific use case.

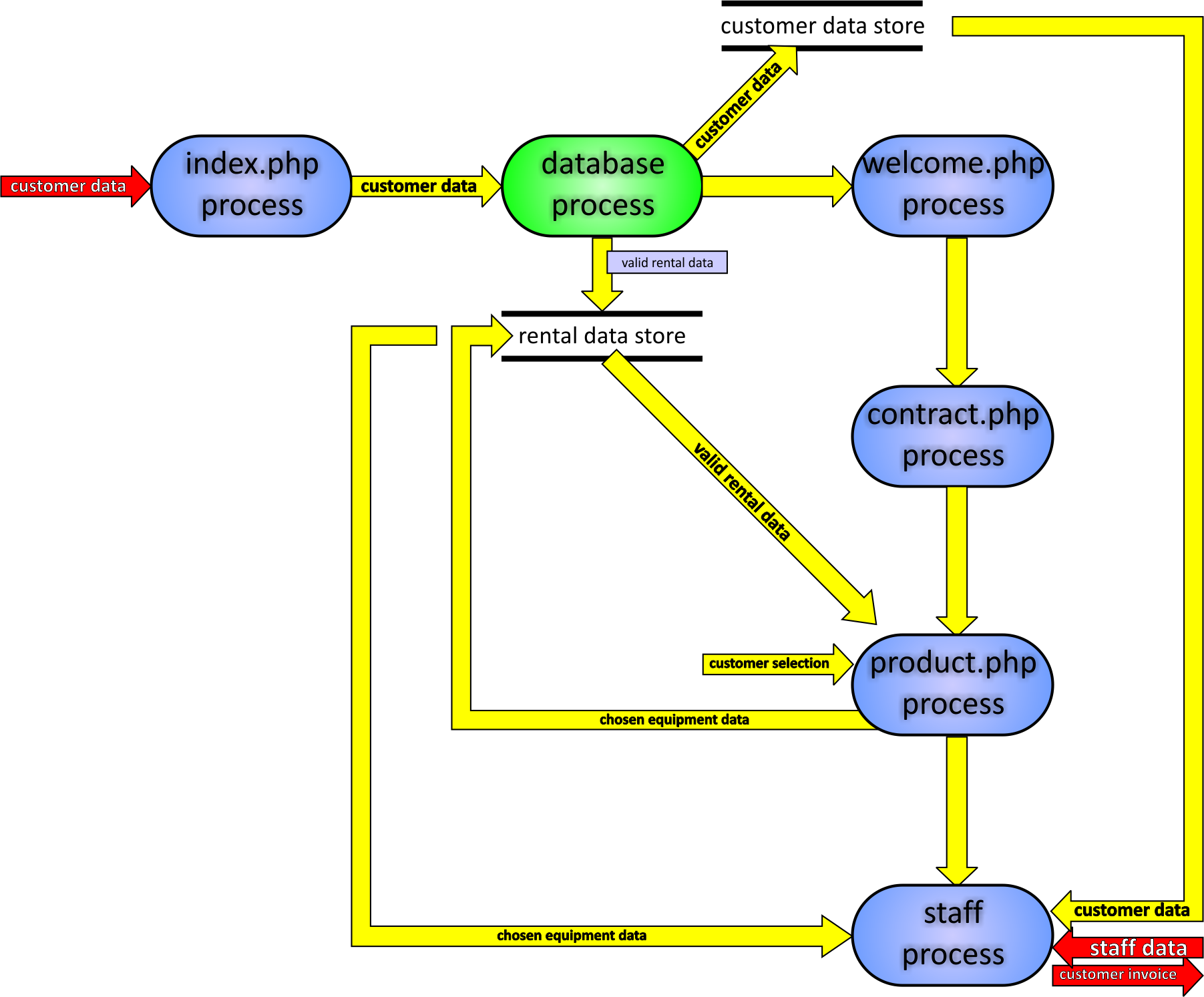
## Customer Rental DFD – Level 0

At the highest level of abstraction, our design of the ACT system is extremely simple when you consider the Level 0 model of the DFD, as is shown in the following figure:



The entire purpose of the system is for the system to stand as an automated intermediary between the customer and the staff personnel. The customer provides data to the system, and then the system processes that data and then ultimately provides the staff with a final invoice representing that customer’s rental transaction. During the process the staff may also input data into the system such as customer end time and so forth.

## Customer Rental DFD – Level 1

At the next level of abstraction, we break down further the “ACT System Process”, as shown:  


Within the ACT System Process, there are additional processes which are responsible for a number of actions within the system, such as – for example – the process titled “index.php” which is a php module. Conceptually, our design dictates that we view index.php as a singular entity which is responsible for three distinct tasks:  
 1. Displaying to the user a web-based Graphical User Interface.  
 2. Receiving input from the user through this GUI.  
 3. Reporting (outputting) the results of the user’s request to the database.

As the figure illustrates, once index.php receives the user’s data, it is forwarded to the database process. The database process is also responsible for several tasks:  
 1. Is the user a valid user? Verify.  
 2. Is the user a new or returning user?  
 3. Did the user provide valid security info (username, password, etc.)  
When the database process verifies the validity of the customer, the database then queries its tables to generate a list of valid rentals based upon the credentials of the customer since some customers may or may not be eligible for some rentals. The database process then forwards this “valid rental data” to a data store we call “rental data store”. In addition, the database process also forwards the customer data to a store called “customer data store”. The data stores can be likened to computer memory; they can be easily accessed (read or write) at any time during the life of the transaction. Finally, the database process will spawn a new process called “welcome.php”.

Here too we view welcome.php as a singular process which takes in input and outputs a result. In this specific instance, the welcome.php process performs 3 specific tasks:  
 1. Displays the customer’s own information so customer sees he is logged in.  
 2. Display’s “class certifications” to customer so customer can review his own status.  
 3. Requests info from customer as to how many guests -- if any -- are with him.  
Our welcome.php process receives information from the customer data store and then performs the three tasks outlined above. Next, welcome.php will spawn a new process called contract.php.

Contract.php displays to the user all of the legal details, liabilities and waivers between the Aquatic Center and its customers. Contract.php requests from the user a simple response (a Boolean yes or no) as to whether or not the user complies with the conditions set forth by the contract. The response from the user is forwarded and saved in the customer data store. Contract.php then spawns the next process: product.php.

The product.php process takes as input the contract response which was saved in the customer data store. If the contract response is false, then at this point it is appropriate to inform the user that he cannot be serviced without agreeing to the contract and then we terminate. This specific use case is not shown. On the other hand, if the contract response is true, then at this point product.php will retrieve the valid rental data from the store and then based on this data it will populate its page with the corresponding products.

The product.php process performs the following tasks:  
 1. Retrieves valid rental data information from the rental data store.  
 2. Displays a GUI which illustrates the equipment available for rent by the customer.  
 3. Allows the user to choose specific equipment desired for rental.  
 4. Allows the user to choose specific quantities of the equipment desired for rental.

The user’s choices of equipment as well as their corresponding quantities represents a new set of data we call “chosen equipment data”. This data is saved in the rental data store.

At this point, from the Aquatic Center employee perspective we have everything that we need: We have the customer’s information in the customer data store and we have their chosen equipment data in the rental data store. Flow of execution then transfers to the process called: “staff process”.  
In the staff process, we have three inputs and one output:  
 1. [in] Chosen equipment data (saved in rental data store)  
 2. [in] Customer data (saved in customer data store)  
 3. [in] Staff data (provided by Aquatic Center Staff; includes staff info, customer end- time, etc.)  
 4. [out] Customer invoice (may include billing, time spent, reports, possible graphs, etc.)  
The staff process reads the chosen equipment data from the rental data store and also the customer data from the customer data store. When the staff process receives its third input (the staff data which includes staff information, customer end-time, any possible discounts, etc.), the staff process will then begin grinding the necessary calculations to produce an invoice will include such things as billing (the final cost of service which the customer must pay), the amount of time the customer spent, an update of the type of equipment that was used and what has thus far been the most popular pieces of equipment, and a host of other reporting services which would be useful to either the staff or admin of the Aquatic Center.

This concludes the level 1 DFD design of the ACT system. The next section outlines the “Level 1 Pseudocode” which – based on the design just described – can be used as a roadmap toward the implementation of the ACT System Process.