Project Cloudbase

Software Architecture Document

Version 3.0

16 April 2015

Harryson Tun, Joshua Greer, Michael Mastantuono
Hood College, Department of Computer Science, Frederick MD, 21701

Revision History

Date	Version	Description	Author
2/20/2015	1.0	Original SA Document	Team Cloudbase
4/16/2015	2.0	Document ready for submission	Team Cloudbase

Document Approval

The following Software Architecture Document has been accepted and approved by the following:

Signature	Printed Name	Title	Date

Table of Contents

1 Introduction

1.1 Purpose

The Software Architecture Document (SAD) contains the architectural description of Cloudbase developed by Joshua Greer, Michael Mastantuono, and Harryson Tun. This description consists of various architectural views of the system, in order to highlight and elaborate the different system aspects and their significance.

The description makes use of the well-known 4+1 view model.

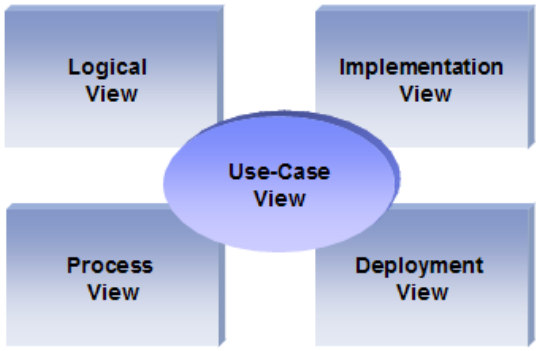

Figure 1.1.1: 4+1 View Model

The 4+1 view model (illustrated in figure 1.1.1) enables various stakeholders to establish the impact of the chosen architecture from their own perspective. Because data is an integral component of Cloudbase, this document will also present a data view.

1.2 Stakeholders

This SAD has two groups of stakeholders: Team Cloudbase and the Mid-Atlantic Soaring Association (M-ASA). Team Cloudbase consists of the system architects Joshua Greer, Michael Mastantuono, and Harryson Tun. Team Cloudbase is developing the system Cloudbase for M-ASA. M-ASA is a non-profit aviation organization that specializes in engineless aircraft also known as gliders.

The M-ASA organization provides four services: aerotow, plane rental, gliding, and simulated rope breaks. For the aerotow service, M-ASA will tow an aircraft from one airport location to another. The plane rental service is self-explanatory: customers can rent a plane. The gliding service is the most common service rendered; engined aircraft tow a glider up to a predetermined altitude and then release it. A simulated ropebreak is a service in which participants simulate an engined aircraft releasing a glider prematurely.

1.3 Definitions, Acronyms, and Abbreviations

Table 1.3.1 contains descriptions for terms, acronyms, and abbreviations that will be used throughout this document.

Term	Description

Admin	Administrator
AOD	Assistant Operations Director
Cloudbase	The software product
CSS	Cascading Style Sheets
DBMS	Database Management System. A software suite that allows a user to create, modify and access databases.
GPL	GNU General Public License version 3 is a copyright license. Among its restrictions, the license requires that all licensed products convey the source code in a public manner. Anyone is free to modify the source code provided all modifications are also licensed under the GPLv3 and therefore made available to the public.
GUI	Graphical User Interface. The part of an application that allows a user to interact with an application
HTML	HyperText Markup Language
LAMP	Linux Apache MySQL PHP/Perl/Python. This is a software stack which is commonly used to host web based applications.
PDF	Portable Document Format
M-ASA	Mid-Atlantic Soaring Association
MySQL	A relational database software product. SQL is an acronym for Structured Query Language.
OD	Operations Director
PHP	PHP: Hypertext Preprocessor is a server-side scripting language
Soaring	The act in which an engineless aircraft is towed into the air and let to glide back down to the ground.

Table 1.3.1: Definitions, Acronyms, Abbreviations

1.4 References

Table 1.4.1 contains a list of references that facilitate building this document.

Source	Website	Description
Mid-Atlantic Soaring Association (M-ASA)	www.midatlanticsoaring.org	M-ASA provides information about its soaring organization

Open Web Application Security Project (OWASP)	www.owasp.org	OWASP provides information and resources about developing a secure web application.

Table 1.4.1: References

1.5 Overview

Table 1.5 gives a preview of the objectives for the remaining chapters of this document.

Section	Reader	Objective
2. Architectural	Software Architect	Overview of architecturally relevant requirements.
3. Logical View	Developer	Knowledge of the application's conceptual structure, as a basis for technical designs.
4. Implementation View	Developer	Knowledge of the application's technical structure.
5. Process View	Developer	Knowledge of the application's run-time behavior.
6. Data View	Developer	Knowledge of the application's database structure.
7. Deployment View	System Administrator roles	Knowledge of the way in which the application is deployed and (internal and external) communication takes place.

Table 1.5.1: Document Structure

2 Architectural Requirements

2.1 Non-functional requirements

Table 2.1.1 contains a list of architecturally significant non-functional requirements. This list is by no means exhaustive. It includes those requirements that influence the system architecture.

Name	Architectural Relevance	Addressed in:
User Authentication	The product shall require authentication in order to progress past the login page.	3.3.1
Secure Password Storage	The product shall store passwords in a secure manner by combining user passwords with a	3.3.1

	randomly-generated salt and then storing the hashed result.	
Data Persistence	Data persistence will be maintained by using a relational database.	3.1, 6
Input	The product shall validate data on input to ensure errors do not arise as a result of incorrect data types or values. Every field will require appropriate validation.	3.1
Concurrency	The product may be accessed simultaneously by multiple users. The maximum recommended number of users is five.	5.2

Table 2.1.1: Non-functional Requirements

2.2 Use-Case View (Functional Requirements)

Table 2.2.1 contains a list of functional requirements depicted by a use case. The list is not exhaustive; it includes the use-cases that drive the architectural design.

Name	Architectural Relevance	Addressed in:
Add/Edit Flight Sheets	Users will be able to create and edit information about each flight.	3.1
Add/Remove Pilots and Aircraft	Pilots, aircraft, and other information will be available for selection from a drop-down menu. Users will have the ability to edit the items that appear on certain menus. For example, users will have the ability to add a new aircraft for selection when inputting flight records.	3.1
Print Reports	Users will be able to print summary reports. Users can modify the time frame for each report and certain content fields.	3.1
Add/Remove Users	Administrative users will be able to add new users and delete existing users.	3.1
Reset User Password	Administrative users can reset other users' passwords. Administrative users do not have access to the plain-text password.	3.1
Change Profile Password and Information	All users have the option to change their passwords and profile information.	3.1

Table 2.2.1: Use Cases

2.3 Users

There will be two classes of user: admin users and standard users. Admin users will have privileges beyond what a standard user has. An admin user may create new users and reset other users' passwords. Standard users will use the system to input data, print reports and alter their own profile information. Figure 2.3.1 illustrates the distinctions between admin users and standard users with respect to use cases.

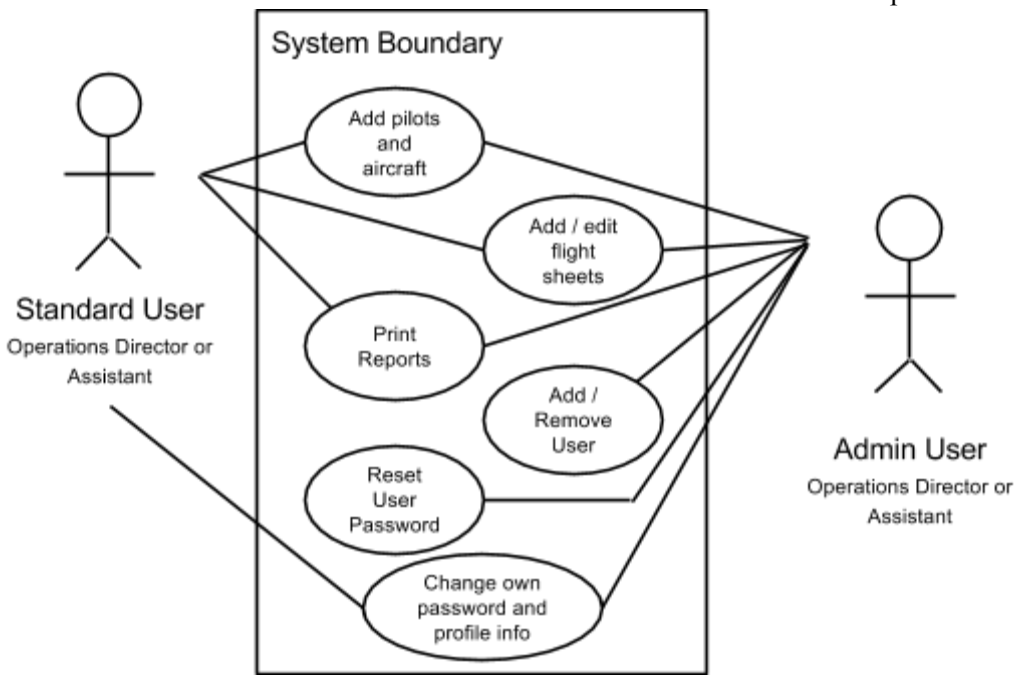

Figure 2.3.1 User Permissions

3 Logical View

3.1 Tiers

The project's logical architecture is based on the three-tier model. Figure 5.1.1 presents a diagram representing Cloudbase's three-tiered architecture.

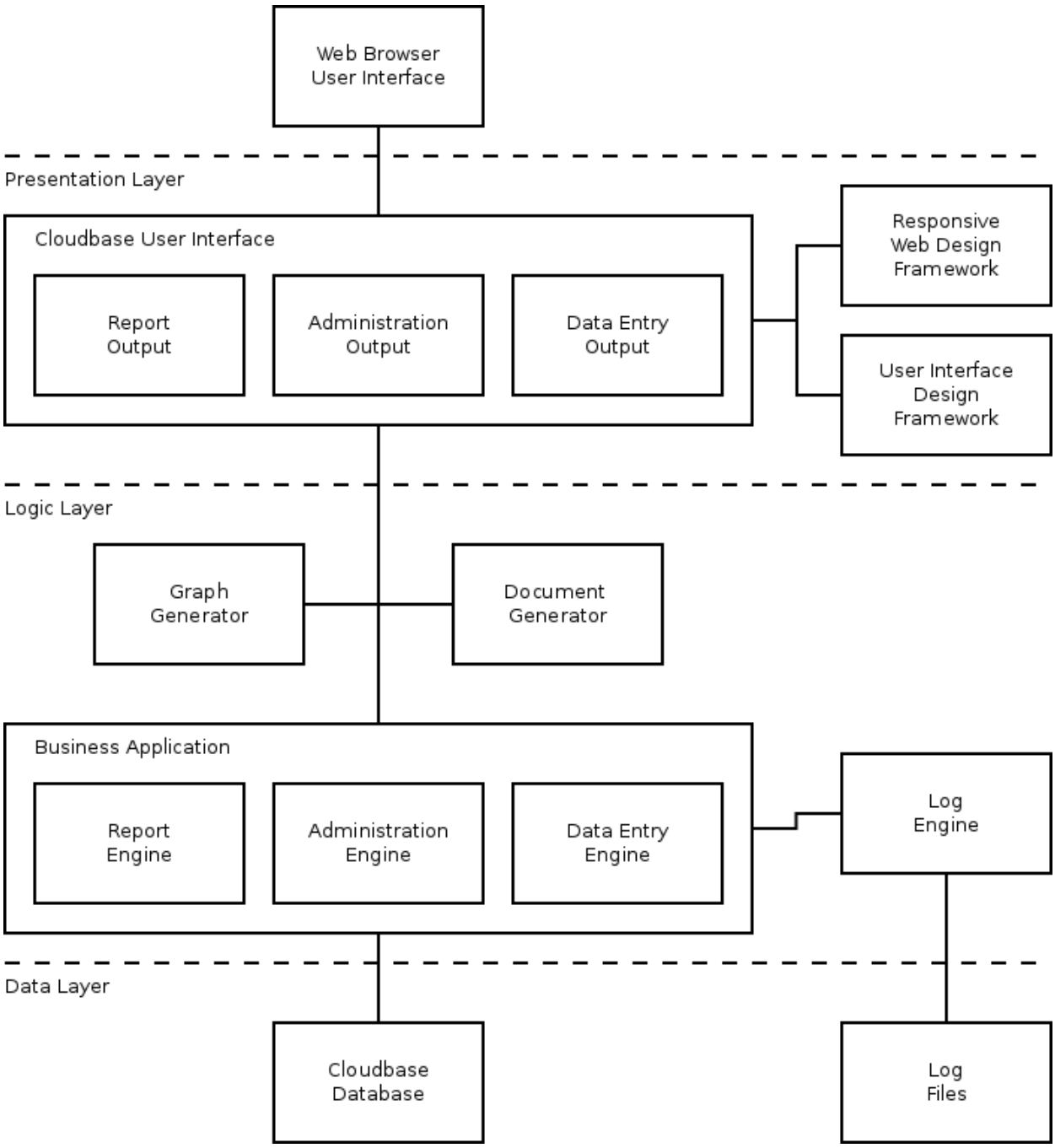

Figure 3.1.1 The Cloudbase Layered-Architecture Model

The three-tier model organizes the system architecture by segregating the system components into three tiers. Two advantages of this model are organization and modularity. The fundamental idea behind the layered-architecture model is that layers are replaceable and therefore modular. This document will make no distinction between layer and tier, though semantically inaccurate.

The three layers are presentation, logic, and data. The names of the layers are synonymous with their scope. The presentation layer is concerned with presenting information to the system users through a

graphical interface. The logic layer consists of business logic components and supplementary libraries. In addition to providing system functionality, the logic layer provides an interface for communication between the data layer and the presentation layer. The third layer is, of course, the data layer. The purpose of the data layer is to store data. Data may be stored in databases or log files.

3.2 Sub-layers

3.2.1 Presentation Layer

Cloudbase's presentation tier consists of three custom components and two libraries. The Cloudbase user interface is composed of the following three custom modules: report output, administration output, and data entry output. Each output module provides output for one of Cloudbase's three major functionalities. Because Cloudbase is a web-based application, each module will ultimately control the presentation of a corresponding web page.

Supplementing Cloudbase's user interface modules are two frameworks: the responsive web design framework and the user interface design framework. The custom Cloudbase user interface modules will use the responsive template in order to provide a responsive web design. The purpose of the user interface design framework is to provide graphical elements such as sortable tables and point-and-click calendar functionality.

3.2.2 Logic Layer

The logic layer will consist of business logic components and facilities for making analytical graphs and PDFs. Again, there are three custom components: Report Engine, Administration Engine, and Data Entry Engine. Each provides business logic operations for one of Cloudbase's three major functionalities. These components serve two purposes. First, the components perform the services required of the business logic such as totaling the flight bill. Second, the components serve as a go-between for the presentation and data layers. The business logic components abstract the database and provide a simplified interface for the presentation layers to obtain information.

Supplementing the business logic components are the document generator and the graph generator. The role of the graph generator is to produce business analytic graphs. The document generator will generate PDFs for print or storage. Although both components produce graphical output, they are tied to the business logic and its interface to the database.

The logic layer will also include a logging engine. Its job is simply to record all system events for testing and debugging purposes.

3.2.3 Data Layer

The data layer consists of a relational database and a log file. The database will contain all business information as well as login credentials. The log file is a simple text file that stores website events. It should be the first stop for collecting information towards debugging.

3.3 Use-Case Realization

The following section describes use cases of sufficient complexity to warrant a closer examination. It also demonstrates communication between logical layers.

3.3.1 Use-Case #1: Login

Figure 3.3.1 illustrates the login process. For security purposes, Cloudbase does not store passwords in plain-text. Instead, Cloudbase stores hashed and salted passwords. To login, the user interface passes the username and password to the business logic layer. At this point the input is filtered by escaping the appropriate characters. The logic layer then checks the database for the given username. If the username is not in the database, the logic layer returns an error to the user interface. If the username exists, the database returns the salt to the logic layer. The logic layer salts the password string and then hashes it. Username and password are sent to the database. If the password is incorrect, an error message is returned to the user interface. If correct, the logic layer informs the user interface that the credentials are a match.

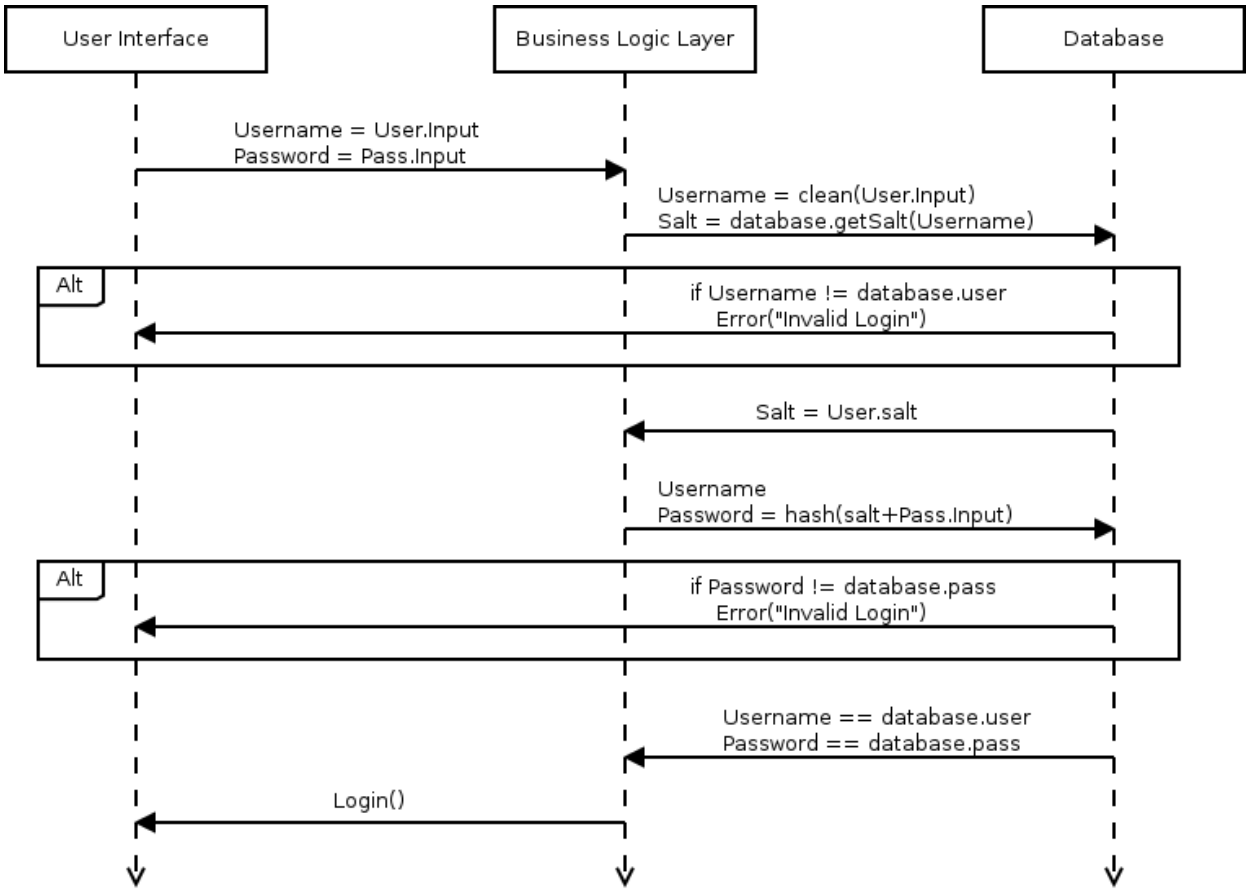

Figure 3.3.1: Login Sequence Diagram

3.3.2 Use-Case #2: Flight Data Entry

Figure 3.3.2 illustrates how Cloudbase will handle concurrent flight input and incomplete data. Once flight #1 takes off, a user enters all available data for flight #1. The data entry processing engine will record the available data in the database and then notify the user, via web interface, that information is missing. Next, the user enters complete flight information. Cloudbase will record the data and then

confirm its storage. Later on, the complete record for flight #1 becomes available. The user will then update the flight #1 database entry to make it complete.

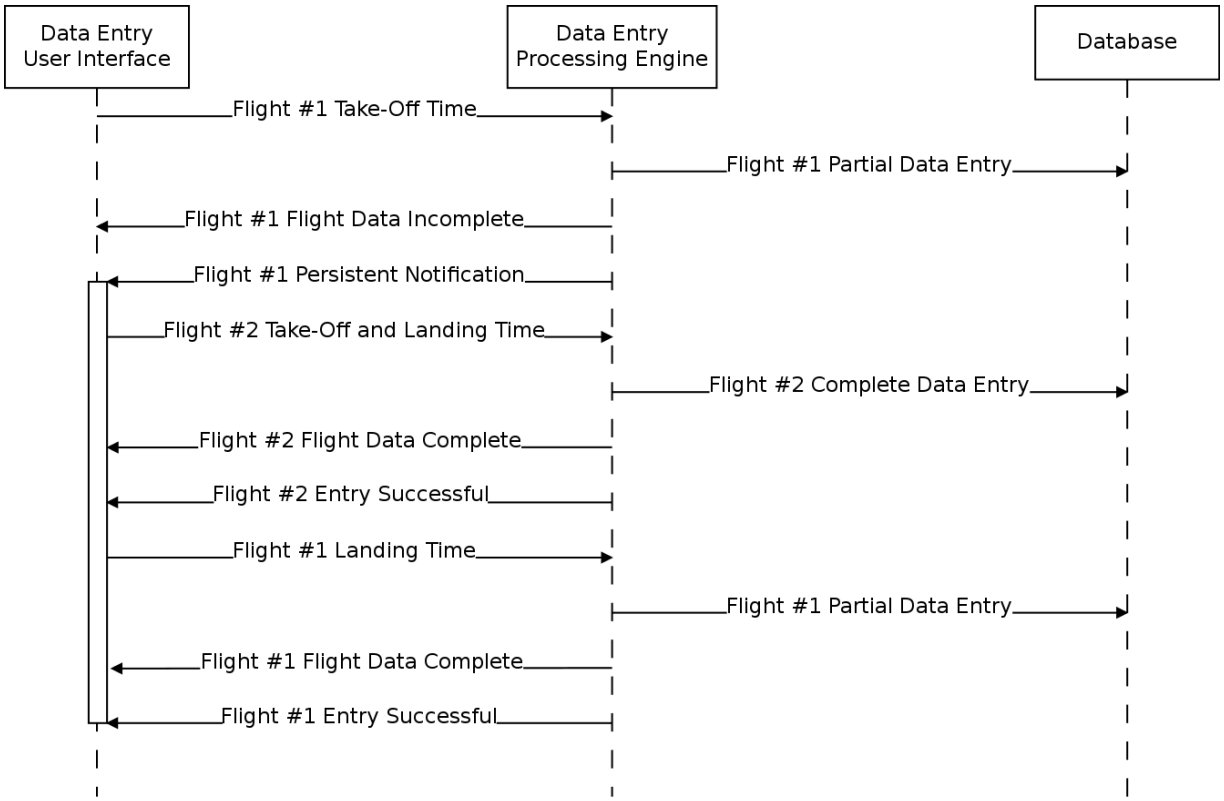

Figure 3.3.2: Flight Data Entry Sequence Diagram

4 Implementation View

4.1 Realization of Tiers

4.1.1 Presentation

The presentation layer consists of all components of Cloudbase that present the graphical user interface. As Cloudbase is a web-based system, these components will ultimately communicate with a web browser. The presentation components may also communicate with business logic components.

The presentation components will be implemented in HTML, CSS, javascript, and PHP. HTML, CSS, and javascript will support the user interface. PHP will primarily serve as the means for communicating with the business logic layer. Libraries for responsive web design and an interactive user interface will supplement the presentation components.

4.1.2 Business Logic

The business logic layer consists of components that bridge the data and presentation layers. These components perform three primary functions. They retrieve data, they store data, and they perform operations on the data. Operations include functional calculations and manipulating data formats.

The business logic layer components will be primarily implemented in PHP, because of the language's capacity for interfacing with a database. Javascript libraries will facilitate making analytic graphs and summary report documents.

4.1.3 Data

The data layer is where all the data are stored. All business data will be stored in the relational database system MySQL. All logged information will be stored in text files.

4.2 (Re)use of components and frameworks

Cloudbase will have a unique user interface and business logic, and therefore many of the components will be built from scratch. However, where possible, Cloudbase will make use of third party components and frameworks. Table 4.2.1 identifies those third party components and describes their functional purpose with relation to Cloudbase.

Component	Description
Chart.js	This is a JavaScript library that produces various types of charts through HTML 5.
jQuery	This is a javascript library. With respect to Cloudbase, it is included because jQuery-UI requires it.
jQuery-UI	This is a javascript library that supports interactive content such as calendar widgets and sortable tables.
TCPDF	TCPDF is a tool for generated PDF documents in PHP.

Table 4.2.1 Third Party Plug-ins

5 Process View

5.1 Overview

Because Cloudbase is a web-based application, it will conform to the client-server architecture model. On the client side, a web browser will be responsible for controlling system processes. On the server side, the web server will be primarily responsible for controlling system processes. The exception to this rule is that Cloudbase will manage concurrent database access.

5.2 Concurrent Database Access

The database will be available for concurrent reading, because there is no conflict. In the event of concurrent updates to the same data, Cloudbase will support a client wins model. In this model, the last changes made to the database are saved.

For example, suppose two users want to update the same flight record at the same time. Also, suppose their updates contain conflicting information. Cloudbase will reconcile the conflict by storing the last update written to the database.

6 Data View

Cloudbase could accurately be described as a user interface for managing a domain-specific database. Therefore, the data view is an integral part of the Cloudbase system. The data view presented in this section is based on the relational database model.

Primary key and foreign key constraints will enforce data consistency and quality. Normalization techniques will minimize data redundancy. Users will primarily access the database through the graphical user interface available on the web. However, system administrators will have access to the underlying relational database software.

Data concerning planes and services are stored in a hierarchical model. This model is akin to an object-oriented model. Common data are pooled in generic data structures from which specialized structures are derived. The advantages to this approach are twofold: data redundancy is reduced, and extensibility is enhanced. New planes and services can be added to the database without modifying the existing data structures.

The remaining parts of this section will survey the data view from three different perspectives: the Entity-Relation model, the relational database schema model, and the data itself.

6.1 Entity-Relation Model

Figure 8.1.1 presents Cloudbase's Entity-Relation model. It is an abstract describintion of a relational database. As its name suggests, the entity-relation model describes organizational entities and the relationships among them. The rectangular boxes represent the entities, and the arrows describe the relationships. The heading to each box is the entity name, and beneath it are the entity's attributes. Dashed lines represent associative entities. Such entities provide additional information about a relationship. For example, when a person purchases a service, a payment is a necessary attribute of that relation. The arrows represent cardinality. For instance, a flight may fulfill one and only one service, but a service may be fulfilled by one or more flights.

The lines without arrows represent descendants. In this case, only service and plane entities have any children. The letter D stands for disjointed. It means the descendants are exclusive entities. A plane, for instance, can not be an engined model and a glider. The descendants are not limited to the ones depicted. New ones can be created.

One noticeable feature of the model in figure 8.1.1 is that the *USER* entity has no lines drawn towards it. The *USER* entity has no relationship with the other entities because it is independent of the organizational rules. It simply stores information concerning website access.

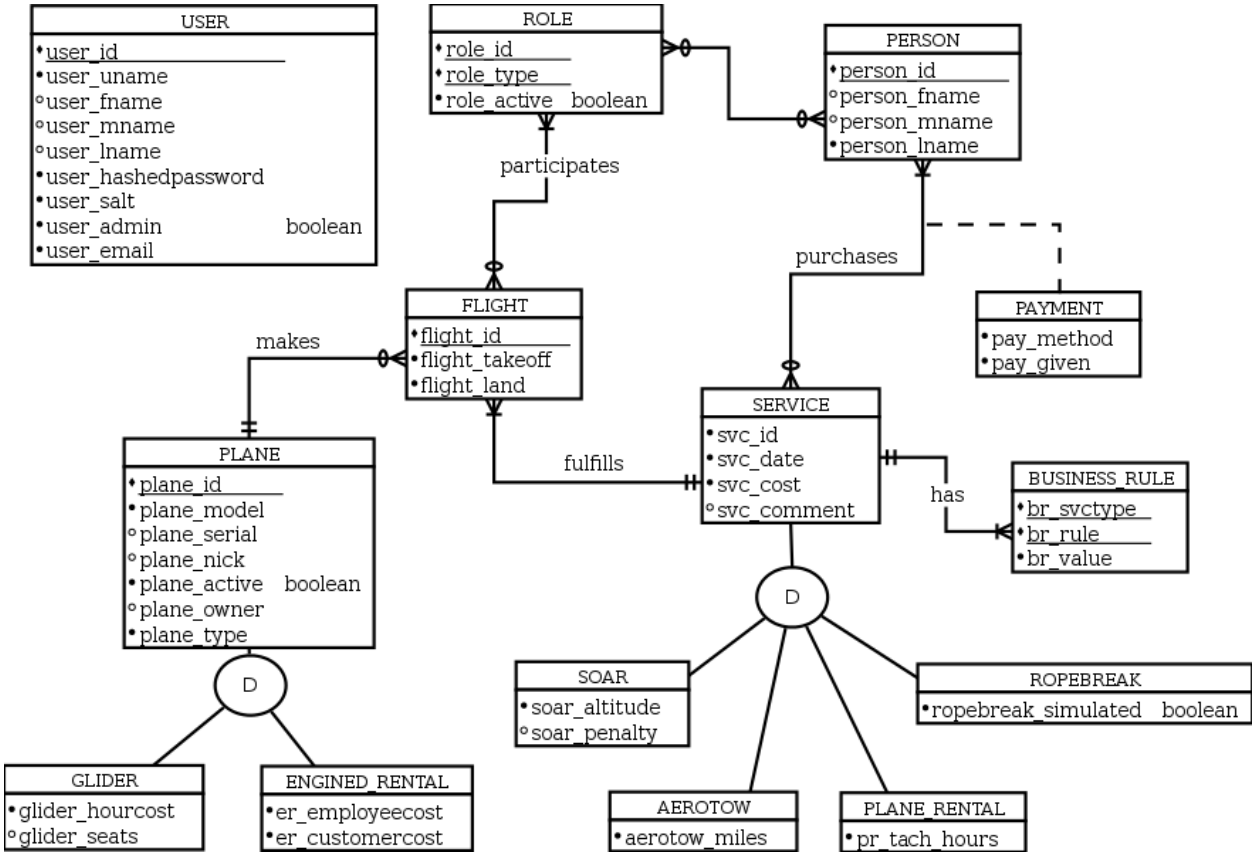

Figure 6.1.1 Cloudbase Entity-Relationship Diagram

6.2 Database Schema Model

One of the advantages of an entity-relation diagram is that it facilitates building a database schema. Cloudbase's database schema is shown in figure 8.2.1. A database schema is more concrete than an entity-relation diagram because it resembles the structure of the database it is intended to represent. Each rectangular cell represents a table row. All adjacent cells form a table. The table name is identified above the leftmost data cell of each table. Solid, underlined rows are the primary keys for each table. Dashed, underlined rows are the foreign keys. The arrows point from foreign keys to the referenced row.

It bears mention that in figure 8.2.1, the *BUSINESS_RULES* entity is now disconnected from the organizational rules. A relationship could be preserved between *SERVICE* and *BUSINESS_RULES* by including an additional table that stored only service types. Because there are only four service types, such a table would introduce an unnecessary level of complexity that is not worth the benefit of foreign key constraints. The database will instead maintain the integrity of service type values with checking constraints.

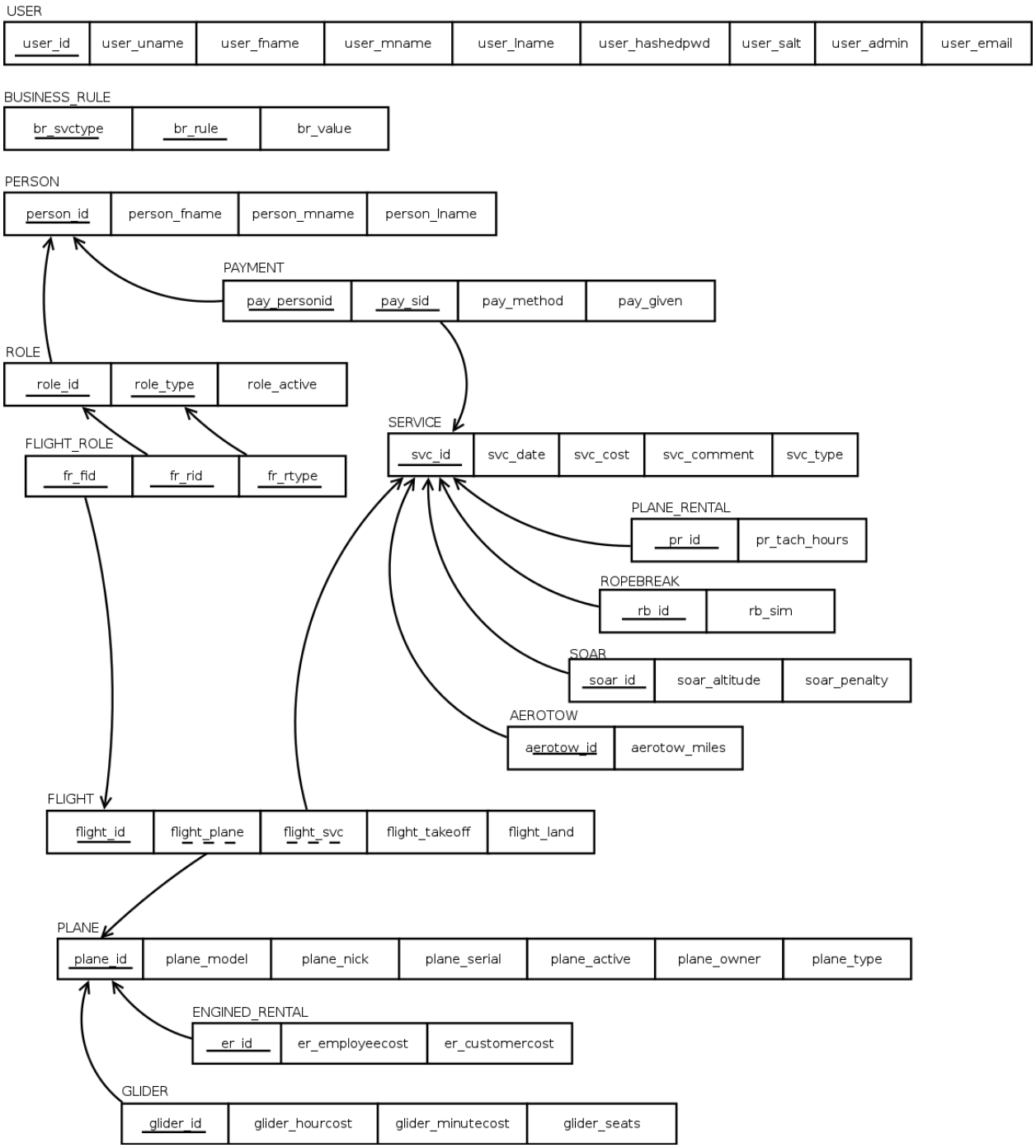

Figure 6.2 Cloudbase Relational Database Schema

6.3 Data Attributes

This section will examine Cloudbase's data view from the most fundamental viewpoint: data attributes. The attributes are organized by database table. Each entry contains a name, value type, and description. A brief explanation will precede every table.

The USER table (6.3.1) will store the necessary information for a system user to log into the system. Of note, the user's password will be stored as a salted hash. The salt will be randomly generated and stored in the USER table. Administrators will have no record of user passwords.

USER		
Value (* Optional)	**Type**	**Description**
user_id	Integer	Each system user will be assigned a unique identification number.
user_uname	String	Username. Must be unique.
user_fname*	String	User's first name
user_mname*	String	User's middle name
user_lname*	String	User's last name
user_salt	String	Randomly generated number
user_hashedpwd	String	The user's password will be combined with the salt and then stored as a hash value. The plain-text password will not be stored.
user_admin	Boolean	Set to true if the user has administrative privileges, false otherwise
user_email	String	It is necessary to store the user's email in order to reset passwords.

Table 6.3.1 User Table

The BUSINESS_RULE table (6.3.2) will contain information concerning the organization's rules. Entries will include the cost of a simulated tow break, the minimum tow height, and base cost, etc.

BUSINESS_RULE		
Value (* Optional)	**Type**	**Description**
br_svctype	String	Identify the type of service to which this rule applies.
br_rule	String	Rule name (e.g. base_altitude_ft)
br_value	String	The value that the rule stores (e.g. 1000)

Table 6.3.2 Business Rule Table

Table 6.3.3 will contain information about the people involved in M-ASA. This table draws no distinction between employees and customers.

PERSON

Value (* Optional)	Type	Description
person_id	Integer	Unique identifying number. Does not match user_id.
person_fname*	String	First name
person_mname*	String	Middle name
person_lname*	String	Last name

Table 6.3.3 Person Table

The purpose of the ROLE table (6.3.4) is to specialize the PERSON table. Because persons have multiple roles, entries will be uniquely identified by person identification number and role. The role_active boolean is intended to identify active employees in order to make drop-down menus with current information.

ROLE		
Value (* Optional)	**Type**	**Description**
role_id	Integer	Matches person_id.
role_type	String	Person's role within the organization (e.g. customer, OD)
role_active	Boolean	This value is only applicable to organizational members. Its purpose is to identify members with active roles. For example, if the OD is still active in the organization, set to true; otherwise, set to false. For non-members, the value should always be false.

Table 6.3.4 Role Table

Table 6.3.5 is a generic table that pools all common service attributes. It is meant to be extended by derivative tables.

SERVICE		
Value (* Optional)	**Type**	**Description**
svc_id	Integer	Every service the organization renders has a unique identification number.
svc_date	Date Object	The date the service was rendered. The month, day, and year will be stored.
svc_cost	Integer	The cost will be stored in cents and as an integer, because floating point values are imprecise.

svc_comment	String	The system user may leave a custom comment pertaining to the customer and/or service.
svc_type	String	The type of service rendered.

Table 6.3.5 Service Table

Table 6.3.6 displays values that apply to the plane rental service.

PLANE_RENTAL		
Value (* Optional)	**Type**	**Description**
pr_id	Integer	This value should match the svc_id of an entry in the SERVICE table.
pr_tach_hours	Float	The cost of the plane rental is dependent on the tachometer hours.

Table 6.3.6 Plane Rental Table

The following table (6.3.7) records both simulated and real ropebreaks.

ROPEBREAK		
Value (* Optional)	**Type**	**Description**
rb_id	Integer	This value should match the svc_id of an entry in the SERVICE table.
rb_sim	Boolean	If the service provided is a rope break, then set this value to true. However, in the event of an accidental rope break, set this to false.

Table 6.3.7 Ropebreak Table

Table 6.3.8 stores values concerning the glider service in which engined planes tow engineless aircraft into the air and then release them.

SOAR		
Value (* Optional)	**Type**	**Description**
soar_id	Integer	This value should match the svc_id of an entry in the SERVICE table.
soar_altitude	Integer	The altitude at which the glider is released.
soar_penalty*	Integer	The penalty is a monetary value stored in cents. This value is optional because the system user may choose not to impose a penalty.

Table 6.3.8 Soar Table

Table 6.3.9 concerns the aerotow service in which a customer's glider is transported from one airport to another.

AEROTOW		
Value (* Optional)	**Type**	**Description**
aerotow_id	Integer	This value should match the svc_id of an entry in the SERVICE table.
aerotow_miles	Integer	The distance an aircraft is towed if this service is provided.

Table 6.3.9 Aerotow Table

Table 6.3.10 contains all data common to all aircraft. The plane_active boolean is meant to keep drop-down menus up to date with current information.

PLANE		
Value (* Optional)	**Type**	**Description**
plane_id	Integer	Each plane will have a unique identification number.
plane_model	String	The airplane model
plane_serial*	String	The airplane's serial number.
plane_nick*	String	The nickname of the plane, if it has one.
plane_active	Boolean	Set to true if the plane is still in active use for the organization; false otherwise.
plane_owner*	String	This is to distinguish between organization-owned planes and privately-owned planes.
plane_type	String	This value is meant to delineate the plane's function (e.g. glider). It will assist in identifying related tables. However, this value is not limited to the definitions of the related tables.

Table 6.3.10 Plane Table

Table 6.3.11 contains information about the engined planes that M-ASA offers for rent.

ENGINED_RENTAL		
Value (* Optional)	**Type**	**Description**

er_id	Integer	This value should match the plane_id of an entry in the PLANE table.
er_employeecost	Integer	The cost, in cents, for a member of the organization to rent this plane.
er_customercost	Integer	The cost, in cents, for a non-member to rent this plane.

Table 6.3.11 Engine Rental Table

Table 6.3.12 contains information about the engine-less aircraft that M-ASA offers for rent.

GLIDER		
Value (* Optional)	**Type**	**Description**
glider_id	Integer	This value should match the plane_id of an entry in the PLANE table.
glider_hourcost	Integer	The cost, in cents, to rent this glider for an hour.
glider_minutecost	Integer	The cost, in cents, to rent this glider for a minute.
glider_seats	Integer	The number of seats the glider has.

Table 6.3.12 Glider Table

Table 6.3.13 may be the single most important table in the whole database. Every flight must be recorded in this database in order to schedule regular maintenance. This table also records the service performed by the flights.

FLIGHT		
Value (* Optional)	**Type**	**Description**
flight_id	Integer	Every flight will have a unique identification number.
flight_plane	Integer	The identification number for the plane in flight.
flight_svc	Integer	The identification number for the service provided by this flight.
flight_takeoff	Time Object	The time at which the plane takes off.
flight_land	Time Object	The time at which the plane lands.

Table 6.3.13 Flight Table

The *FLIGHT_ROLE* table is to identifies the people, and their roles, involved in each flight. Roles may include OD, customer, pilot, instructor, etc.

FLIGHT_ROLE		
Value (* Optional)	**Type**	**Description**
fr_fid	Integer	The identification number for the flight in which this person was involved.
fr_rid	Integer	The identification number for this person
fr_rtype	Integer	The role of this person in this flight.

Table 6.3.14 Flight Role Table

Table 6.3.15 records information about payment for services rendered.

PAYMENT		
Value (* Optional)	**Type**	**Description**
pay_personid	Integer	The person_id of the person providing payment.
pay_sid	Integer	The svc_id of the service for which this person is paying.
pay_method	String	The method by which this person is paying. The values are informed by organizational rules and may include VISA, check, etc.
pay_given	Integer	The amount this person paid in cents.

Table 6.3.15 Payment Table

7 Deployment View

Cloudbase is designed to operate on top of a LAMP stack, a server environment consisting of Linux, Apache, MySQL and PHP. While there are many software configurations that claim to reproduce the functionality of LAMP software, this system will only be tested on a LAMP stack. Alternative software configurations may very well run Cloudbase without any problem, but it is only guaranteed to work on a LAMP stack.

For full web-accessibility, the system requires an internet-connected host server. Because the system is designed for a maximum of five simultaneous users, Cloudbase requires only modest processor, memory, and hard-disk performance. This system is not designed to scale from a few users to hundreds or even thousands of users.

7.1 Server Side Requirements

The product will run on any version of MySQL at or above version 5.6. The product will run with any version of PHP at or above version 5.6.5. The product will run on any version of Apache at or above version 2.4.12.

7.2 Security

To increase the security of the system, Cloudbase should be installed on a trusted platform that supports HTTPS connections and a configurable firewall. HTTPS will encrypt communications between host and client. Because Cloudbase is designed for a local organization, a firewall configuration that drops international traffic is recommended.

7.3 Client Side Requirements

The product must be accessible and function properly on Internet Explorer 11+, FireFox 35+, Google Chrome 40+ and Safari 7+. The product will be optimized for use on tablets and mobile devices.