Database Project

Architecture/Design Document

**Table of Contents**

**1 Introduction.. 2**

**1.1 Database Arch/Dependencies..2**

**2 Design Goals.. 3**

**2.1 U.I. Design Goals.. 3**

**3 System Behavior.. 4**

**4 Logical View... 5**

**4.1 High-Level Design (Architecture) 5**

**4.2 Mid-Level Design. 5**

**4.3 Detailed Class Design. 6**

Change History

**Version:** 1.0

**Modifier:** Nick King

**Date:** 3/17/2017

**Description of Change:** Baseline

# **1** **Introduction**

**Architecture and Design**

The purpose of this architecture/design document is to explain the organization of the code in an effort to make it easier for new programmers to become familiar with the code. This architecture/design document will identify major system components and describe their static attributes and dynamic patterns of interaction.

*This document describes the architecture and design for the CS 470 Databases Project application being developed for users that desire a simple representation of their budgeting options. This software will provide graphical interpretations of spending, and notify the user by email when they are about to exceed their budget for a given time frame.*

The architecture and design for a software system is complex and individual stakeholders often have specialized interests. There is no one diagram or model that can easily express a system’s architecture and design. For this reason, software architecture and design is often presented in terms of multiple views or perspectives [IEEE Std. 1471]. Here the architecture of the Database Project application is described from 4 different perspectives [1995 Krutchen]:

1. Logical View – See [4.2] Mid-Level Design
2. Process View – GUI Thread, PageLoad, Postback
3. Development View – See [4.2] future edits. See how system modules map to development organization.
4. Use Case View – [TBD] the use case view is used to both motivate and validate design activity. At the start of design the requirements define the functional objectives for the design. Use cases are also used to validate suggested designs. It should be possible to walk through a use case scenario and follow the interaction between high-level components. The components should have all the necessary behavior to conceptually execute a use case.

**1.1 Database Architecture and Dependencies**

***n-Tiered Architecture- see [4.1]***

*CS 470 Database Project is a multitiered application that has client and server components, a database repository, and a given company's network infrastructure, which represents the targeted resources to be managed by the software. [1]*

*The database will be designed using Microsoft SQL Server 2016, along with SQL Server Management Studio.*

*The web application will be developed in Visual Studio 2015 Community Edition using ASP .NET framework, with the MVC architecture for coherent application control logic.*

The Database has a few constraints and dependencies:

* Accounts must be attached to a user
* Budgets and Transactions must be attached to an account
* Budgets must have a single specific category
* Budgets are dependent on the categories table to resolve category names
* Budgets are dependent on accounts to resolve transactions

# **2** **Design Goals**

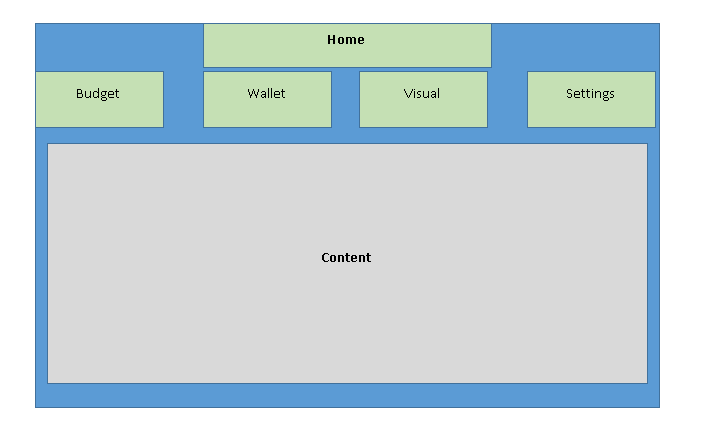
There is no absolute measure for distinguishing between good and bad design. The value of a design depends on stakeholder priorities. For example, depending on the circumstances, an efficient design might be better than a maintainable one, or vise versa. Therefore, before presenting a design it is good practice to state the design priorities. The design that is offered will be judged according to how well it satisfies the stated priorities.

The design priorities for the CS 470 Database application are, in no desired order:

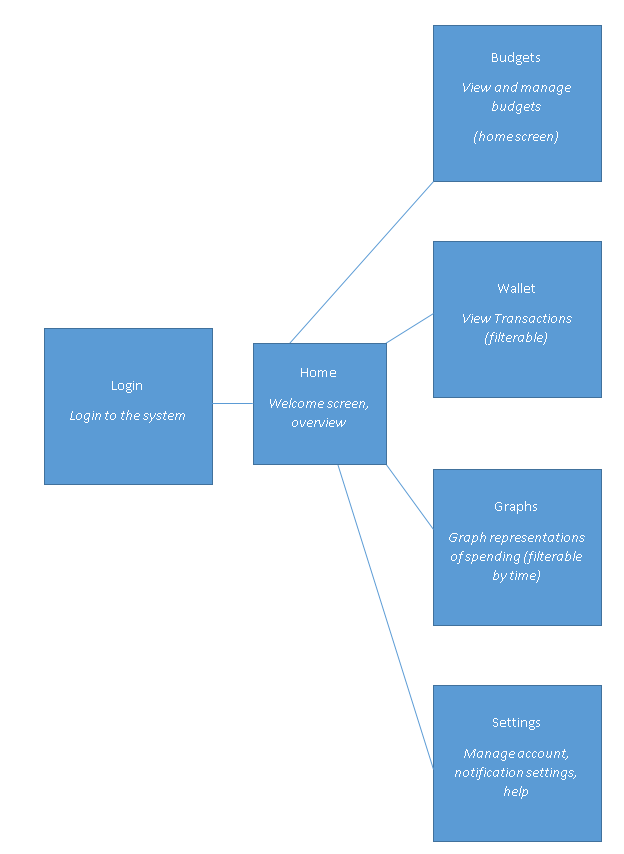
* The design should minimize complexity and development effort.
* The design will implement desirable normal form database schema.
* The design should allow for simplicity in the integration of new modules and features.
* The design will address the requirements listed in the project deliverables.
* The design will prioritize simple modules to be used by budgeting features.

**2.1 U.I. Design**

This project will utilize html, css, javascript, jquery, and bootstrap to present a reactive and simple user interface that displays user budgeting information modally in order to be aesthetically pleasing. The user interface will not contain false allowances, and will use standardized design formulae to display meaningful data to the user that is interactive.



# **3** **System Behavior**

The use case view is used to both drive the design phase and validate the output of the design phase. The architecture description presented here starts with a review of the expected system behavior in order to set the stage for the development process

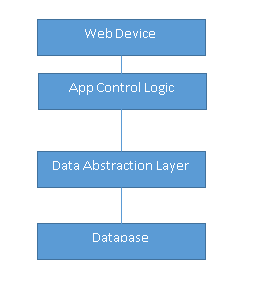
# **4** **Logical View**

The logical view describes the main functional components of the system. This includes modules, the static relationships between modules, and their dynamic patterns of interaction.

In this section the modules of the system are first expressed in terms of high level components (architecture) and progressively refined into more detailed components and eventually classes with specific attributes and operations.

## **4.1** **High-Level Design (Architecture)**

The high-level view or architecture consists of 4 major components:



|  |  |
| --- | --- |
|  |  |
|  |  |

System Architecture

* The **Web Connected Device** provides the user with an interface to interact with the application
* The **Database** is a central repository for data on users, accounts, and other necessary data for budgeting tools
* The **Database Abstraction Layer** is the collection of functions to query the database-- it responds to calls from the Application Control Logic layer..
* The **Application Control Logic** is the main driver of the application. It presents information to the user and reacts to user inputs.

## **4.2** **Mid-Level Design**

See ER class diagram

Class Diagram will be edited as new classes are implemented

## **4.3** **Detailed Class Design**

--webforms include the abstract layer and create Data abstract objects that allow for the database to be queried through the webform. This data allows for the webform logic to manipulate the U.I. through scripts. The backbone logic is largely built, the front end logic is still being implemented.

The webforms use a DataAbstract object to call these functions.

*More detailed interaction:*

public DataSet get\_Transactions(int acct\_num)  
 {  
 //selects accounts for a specific suer id  
 string selectSQL = "SELECT \* FROM Transactions WHERE AcctNumber = " + acct\_num;  
   
 SqlCommand cmd = new SqlCommand(selectSQL, connect);  
   
 DataSet temp = new DataSet();  
 SqlDataAdapter adapter = new SqlDataAdapter(cmd);  
   
 connect.Open();  
 adapter.Fill(temp);  
 connect.Close();  
 return temp;  
 } //end get\_Transactions

-More details/diagram work to come as features get implemented.

External Sources:

[1] https://docs.oracle.com/cd/B16240\_01/doc/em.102/e14653/get\_started\_acc\_overview.htm