POS SYSTEM – ARCHITECTURE DRIVERS



HIT Team

Consulting

Sales

Staffing

Support

# Information of document

|  |  |
| --- | --- |
| **Title** | **Architecture Driver Document** |
| **Author(s)** | All team |
| **Reviewer(s)** | All team |
| **Team name** | HIT Team |
| **Team members** | Thanh Giang, Hiep Ta, Phuc Nguyen, Giang Nguyen, Dat Tran, Huy Huynh |
| **Project mentors** | Ms. Huong Nguyen, Mr. Huyen Pham |
| **Editor** |  |
| **Type of report** | Architecture Driver Document |
| **Software used** | MS Word |

# Document Reviewer Information

|  |  |  |
| --- | --- | --- |
| Reviewer Name | Review Attendance (R/S) | Comments |
|  |  |  |
|  |  |  |
|  |  |  |

# Document Approver Information

|  |  |  |
| --- | --- | --- |
| Approver Name | Approver Function | Comments |
|  |  |  |
|  |  |  |
|  |  |  |

# Document Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date | Revision | Status | Change Summary | Revised by |
| 6/3/2012 | 1.0 |  | Consume Team member’s tasks | Thanh Giang |
| 15/4/202 | 1.1 |  | Update Quality Attributes | Giang Nguyen |
| 18/4/2012 | 1.2 |  | Usecase | Thanh Giang |
| 24/05/2012 |  | start | Use case Description | Hiep Ta |
| 9/6/2012 |  |  | Module View | Huy Huynh |
| 21/6/2012 | 3.3 |  | Static Perspective & Dynamic Perspective | Huy Huynh |

Contents

[Information of document 2](#_Toc328236038)

[Document Reviewer Information 2](#_Toc328236039)

[Document Approver Information 2](#_Toc328236040)

[Document Revision History 2](#_Toc328236041)

[**1.** **Document description** 14](#_Toc328236042)

[**1.1.** **Purpose and audience:** 14](#_Toc328236043)

[**1.2.** **Document organization:** 15](#_Toc328236044)

[**1.3.** **References and relevant document:** 17](#_Toc328236045)

[**2.** **Project overview** 17](#_Toc328236046)

[**2.1.** **Purpose and Scope** 17](#_Toc328236047)

[**2.2.** **Stakeholders** 18](#_Toc328236048)

[**3.** **Architectural drivers** 18](#_Toc328236049)

[**4.** **System context** 18](#_Toc328236050)

[**5.** **Physic Perspective** 19](#_Toc328236051)

[**5.1.** **Primary presentation:** 20](#_Toc328236052)

[**5.2.** **Element catalog:** 20](#_Toc328236053)

[**5.2.1.** **Elements and their properties** 20](#_Toc328236054)

[**5.2.2.** **Relations and their properties** 21](#_Toc328236055)

[**5.2.3.** **Element behavior** 21](#_Toc328236056)

[**5.3.** **Architecture background** 21](#_Toc328236057)

[**5.3.1.** **Rationale design** 21](#_Toc328236058)

[**5.3.2.** **Analysis of results** 22](#_Toc328236059)

[**5.3.3.** **Assumptions reflected in the design** 22](#_Toc328236060)

[**6.** **Static Perspective** 22](#_Toc328236061)

[**6.1.** **Primary presentation:** 22](#_Toc328236062)

[**6.1.1.** **Layered Style** 22](#_Toc328236063)

[**6.1.2.** **Uses Style** 22](#_Toc328236065)

[**6.1.** **Element catalog:** 23](#_Toc328236066)

[**6.1.1.** **Elements and their properties** 23](#_Toc328236067)

[**6.2.1.1** **MVC** 23](#_Toc328236068)

[**6.1.2.** **Relations and their properties** 25](#_Toc328236069)

[**6.2.** **Context diagram:** 25](#_Toc328236070)

[**6.3.** **Architecture background:** 25](#_Toc328236071)

[**6.4.** **Glossary of terms:** 27](#_Toc328236078)

[**6.5.** **Other information:** 27](#_Toc328236079)

[**7.** **Data Model** 28](#_Toc328236080)

[**8.** **Dynamic Perspective** 30](#_Toc328236081)

[**8.1.** **Primary presentation:** 30](#_Toc328236082)

[**8.2.** **Element catalog:** 31](#_Toc328236083)

[**8.2.1.** **Elements and their properties** 31](#_Toc328236084)

[**8.2.2.** **Relations and their properties** 34](#_Toc328236085)

[**8.2.3.** **Element behavior** 34](#_Toc328236086)

[**8.2.3.1.** **Add Store** 34](#_Toc328236087)

[**8.2.3.2.** **Add Loyal Member** 35](#_Toc328236089)

[**8.2.3.3.** **AddUser** 36](#_Toc328236091)

[**8.2.3.4.** **AddUserComputer** 37](#_Toc328236093)

[**8.2.3.5.** **Statistic** 38](#_Toc328236095)

[**8.2.3.6.** **AddNewCategogy** 39](#_Toc328236097)

[40](#_Toc328236098)

[**8.2.3.7.** **AddNewProduct** 40](#_Toc328236099)

[**8.2.3.8.** **Sale** 41](#_Toc328236101)

[**8.3.** **Architecture background:** 43](#_Toc328236103)

[**8.3.1.** **Rationale design** 43](#_Toc328236104)

[**8.3.2.** **Analysis of results** 43](#_Toc328236105)

[**8.3.3.** **Assumptions reflected in the design** 43](#_Toc328236106)

[**8.4.** **Glossary of terms:** 43](#_Toc328236107)

[**9.** **Solution background** 43](#_Toc328236108)

[**9.1.** **Architectural Approaches** 43](#_Toc328236109)

[**9.1.1.** **Ensuring Performance:** 43](#_Toc328236110)

[**9.1.2.** **Ensuring Performance:** 43](#_Toc328236111)

[**Ensuring operation in POS that should be performed rapidly.** 43](#_Toc328236112)

[**10.** **Mapping between perspectives** 44](#_Toc328236113)

[**10.1.** **Mapping between module view and allocation view.** 45](#_Toc328236114)

[-- The End -- 46](#_Toc328236115)

1. **Document description**
   1. **Purpose and audience:**

This document provides a high level overview of the evolving technical architecture for the Sales System of a retail chain using a loyalty card point system. It also provides a high-level description of the goals of the architecture, the use cases support by the system and architectural styles and components that have been selected to best achieve the use cases.

In addition to these views, this architectural description will:

* Identify the candidate patterns and tactics that will become the architecture design of the system.
* Frame the architectural design activity, begin with the given technical constraints and the structures
* Identify patterns generally suit the needs described in the product description based upon the quality attribute scenarios
* Identify tactics we apply to further refine the initial decomposition of the system and promote the necessary quality attributes.

The architecture has a set of guiding principles as well as known criteria and constraints that shape the proposed architecture. It is intended to capture and convey the significant architectural decisions which have been made on the system.

The development team can use this document to review the architecture of the system. The Architecture document will be also useful for future development teams.

* 1. **Document organization:**

Sub-sections of Section 1 include the following.

* Section 1.1: Purpose and audience: Describe who the intended audience and organizations are and what they might use the document for.
* Section 1.2: Document organization: Describe the overall organization of the document. List the major sections of the document and describe what concerns each section addresses.
* Section 1.3: Common notation: List any notation that will be used throughout the document.
* Section 1.4: Terminology and definitions: Define any terms used throughout the document and provide context for terminology.
* Section 1.5: References and relevant document: List any other relevant documents that the reader might need to refer to, and most importantly, describe their relationships to this document and why the reader might want to (or need to) refer to them.

Sub-sections of Section 2 include the following.

* Section 2.1 and 2.2: This section describes the project and its purpose and scope, why the system is being built.
* Section 2.3: List the relevant stakeholders, their organizations, and how they will interact with the system.

Sub-sections of Section 3 include the following: *In this section describe the architectural drivers for the system.*

* Section 3.1: Use-case diagram of the system and list the priority of the use-case
* Section 3.2: Business Constraints and Technical Constraint of the system and list the priority of these constraints
* Section 3.3: Quality Attributes and Technical Constraint of the system and list the priority of these Quality Attributes

Sub-sections of Section 4 include the following: The system context is the first step in design and should include at least one context drawing. In addition to the context drawing, we show the scope of the system being described by showing its relationship to external entities like systems, peripherals, organizations, and stakeholders as necessary to describe the context drawing. We also describe the relative perspective of the context drawing or drawings.

Sub-sections of Section 5-6-7-8-9 include the following: specify the software architecture. Views specify elements of software and the relationships between them. A view corresponds to a viewpoint and is a representation of one or more structures present in the software

Sub-sections of Section 10 include the following, which is an index of architectural elements and relations telling where each one is defined and used in this SAD. The section also includes a glossary and acronym list.

* 1. **References and relevant document:**

|  |  |
| --- | --- |
| **Name** | **Description** |
| **Sales System of a retail chain using a loyalty card point system** | System description |
| **Architecting software intensive systems** | A practitioner’s guide, Anthony J. Lattanze |

1. **Project overview**
   1. **Purpose and Scope**

The project will aim to develop a sale system for Company A, a retail chain (hereinafter, the system) in conjunction with its launch of a point service.

The system consists of a main server, located at the head office. The head office server and the POS terminals are connected to each other via a network. Products sold at stores have bar codes attached which indicate the product codes. These bar codes can be read with bar code readers of POS terminals. Customer who have become point service members are issued point cards, which bear bar codes indicating their member numbers, and when they purchase products with cash, they are awarded points based on the amount of their purchase.

For each product, its standard price, common to all stores, is set as a part of the product data. Each store, however, can set and use its own actual retail price instead of the standard price during the limited period specified is each store. The actual retail price must be set in advance, and it cannot be charged in the middle of the specified period.

Products are classified into product types such as food, general merchandise, etc. Not all stores carry every product type, and the range of product types carried is designated for each store.

Customer also use online website at everywhere to check their information such as personal information, loyal point.

* 1. **Stakeholders**

|  |  |  |
| --- | --- | --- |
| **Stakeholders** | **Organizations** | **Interaction** |
| **Customers** | None | Can check their point at online store website |
| **Staffs** | Company A | Responsible for manage information of products, categories, customer, |
| **Cashiers** | Company A | Responsible for check bills (records) |
| **Manager** |  | Responsible for manage retail stores, and they can statistic sales by many criterion |
| **Administrator** | Company A | Responsible for manage user of the system such as: Create new, assign authorize. He can also manage computer of user computer so they can access while they are not working |

1. **Architectural drivers**

***Please reference to file “Architectural Driver”***

1. **System context**



**Users and roles:** Stakeholders who interact to Retail system was described in section 2.3.Stakeholder of this document.

**Channels:** Users will use different channels to access the system.

* Staff: Use website on PC locates at Retail Store to access to system.
* Manager: use website on everywhere PC to access to system.
* Cashier: Use Bar code reader at computer using store website to interacts with system
* Customer: Use online website to see their point
* Administrator: Use PC at Head Office to access the system

**Relationship Describe:**

The context diagram shows the input of stakeholders and output from system, direction of the arrows show the direction of information.

: Show that the input from user to the system

: Show that the output from system to the user

1. **Physic Perspective**

**ALLOCATION VIEW**

**(Deployment Style)**

* 1. **Primary presentation:**



* 1. **Element catalog:**
     1. **Elements and their properties**

.

|  |  |  |  |
| --- | --- | --- | --- |
| Associated Drawings:  Fig2 | | | Perspective:  Physic |
| No | **Name** | **Properties** | **Responsibilities** |
| 1 | Main Database | * It‘s a database run in database management system SQL server 2008 | * Contains all general data of system |
| 2 | Backup database | * It‘s a database run in database management system SQL server 2008 | * Contains all general data of system. Run parallel with the main database |
| 3 | WEB POS Application | * It’s a WEB application |  |
| 5 | Database Server | * OS: Windows Server 2008 * Processor: 1 x Intel® Xeon® Processor E5606 * Memory: 1 x 2GB DDR3 1333 240- * Hard Disk: DELL 250GB SATA 7.2K 3.0Gbs 3.5" Enterprise * Software: Microsoft SQL Server 2008 Enterprise, .NET Framework 4.0 | * Run Main Database * Run Backup Database |
| 6 | WEB server | * Software: IIS 7.0 |  |
| 6 | User PC | * Operation System : Genuine Windows® 7 Home Basic, * Processor: AMD AM3 For Phenom™ II/Athlon™ II Family /Processors * Chipset: AMD SB710 * Graphics: ATI Radeon HD 3200, * Hard Drive SATA: 3.5" 320G * Software: Web browser | * Head office and Retail Store   + Run WEB POS Application |
| 7 | Fiber router | * Vigor2950 | * Connect LAN with WAN |
| 8 | LAN | * Topology: Star * Use switch to connect elements in LAN | * Connect computers in a store |
| 11 | Fiber cable |  | * Connect fiber router in WAN by fiber port |

* + 1. **Relations and their properties**
    2. **Element behavior**
  1. **Architecture background**
     1. **Rationale design**

Deployment design satisfies quality attributes following:

* + Performance: there is main data base is stored at head office, so request from Retails Store will be very fast because it access directly to the main database.
    1. **Analysis of results**
    2. **Assumptions reflected in the design**

1. **Static Perspective**

**Module View**

**(Layered Style and Uses Style)**

* 1. **Primary presentation:** 
     1. **Layered Style**



* + 1. **Uses Style**



* 1. **Element catalog:**
     1. **Elements and their properties**
        1. **MVC**

|  |  |  |
| --- | --- | --- |
| Elements | | Properties |
| Controller | **ProductControllers** | The ProductController class contains action methods that render view pages (AddProduct, EditProduct, ViewProduct) |
| **Category**  **Controllers** | The CategoryController class contains action methods that render view pages (AddCategory, EditCategory, ViewCategory) |
| **StatisticsControllers** | The StatisticsController class contains action methods that render view pages (Statistics) |
| **SaleControllers** | The SaleController class contains action methods that render view pages (BillManagement, Checkout, PriceLog) |
| **StoreControllers** | The StoreController class contains action methods that render view pages (AddStore, ViewStore, EditStore) |
| **LoyalMember**  **Controllers** | The LoyalMemberController class contains action methods that render view pages (AddLoyalMember, ViewLoyalMember, EditLoyalMember) |
| **StoreCategory**  **Controllers** | The StoreCategoryController class contains action methods that render view pages (AddStoreCategory, ViewStoreCategory, EditStoreCategory) |
| **ComputerControllers** | The POSController class contains action methods that render view pages (AddPOS, ViewPOS, EditPOS) |
| **UserControllers** | The UserController class contains action methods that render view pages (AddUser, ViewUser, EditUser) |
| View | **AddStore** | This GUI helps the user to add a new Store. |
| **ViewStore** | This GUI helps the user to view the Store. |
| **EditStore** | This GUI helps the user to modify some information about the Store. |
| **AddCategory** | This GUI helps the user to add a new Product Category. |
| **ViewCategory** | This GUI helps the user to view the Product Category. |
| **EditCategory** | This GUI helps the user to modify some information about the Product Category. |
| **AddProduct** | This GUI helps the user to add a new Product. |
| **ViewProduct** | This GUI helps the user to view the Product. |
| **EditProduct** | This GUI helps the user to modify some information about the Product. |
| **BillManagement** | This GUI helps the user to view the list of Bills and the user can see some detail information of each Bill. |
| **PriceLog** | This GUI shows all Price history that was used for each product |
| **Checkout** | This GUI helps the user check bill and make a payment. |
| **AddUser** | This GUI helps the user to add a new User. |
| **ViewUser** | This GUI helps the user to view the User. |
| **EditUser** | This GUI helps user to modify some information about the User. |
| **AddLoyalMember** | This GUI helps the user to add a new loyal Customer. |
| **ViewLoyalMember** | This GUI helps the user to view the loyal Customer. |
| **EditLoyalMember** | This GUI helps user to modify some information about the loyal Customer. |
| **AddStoreCategory** | This GUI helps the user to add a new Retail Store Category. |
| **ViewStoreCategory** | This GUI helps the user to view the Retail Store Category. |
| **EditStoreCategory** | This GUI helps user to modify some information about the Retail Store Category. |
| **AddComputer** | This GUI helps the user to add a new Computer. |
| **EditComputer** | This GUI helps user to modify some information about the Computer. |
| **ViewComputer** | This GUI helps the user to view the Computer. |
| **Statistics** | To make statistics about the total amount of product (or product category) was bought on month. |
| Model | **ProductModels** | Storing and retrieving the Product information and return a message back to view pages (AddProduct, EditProduct, ViewProduct) |
| **SaleModels** | Storing and retrieving the Sale information and return a message back to view pages (BillManagement, Checkout, PriceLog) |
| **StoreModels** | Storing and retrieving the Retail Store information and return a message back to view pages (AddStore, EditStore, ViewStore) |
| **LoyalMember**  **Models** | Storing and retrieving the Customer information and return a message back to view pages (AddLoyalMember, EditLoyalMember, ViewLoyalMember) |
| **UserModels** | Storing and retrieving the Userinformation and return a message back to view pages (AddUser, EditUser, ViewUser) |
| **CategoryModels** | Storing and retrieving the Product Categoryinformation and return a message back to view pages (AddProductCategory, EditProductCategory, ViewProductCategory) |
| **StoreCategory**  **Models** | Storing and retrieving the Store Category information and return a message back to view pages (AddStoreCategory, EditStoreCategory, ViewStoreCategory) |
| **ComputerModels** | Storing and retrieving the POSinformation and return a message back to view pages (AddPOS, EditPOS, ViewPOS) |
| **StatisticsModels** | Storing and retrieving the some information related Statistics and return a message back to view pages (Statistics) |
| **DBContext** | Context class is the primary class for interacting with data as objects that are instances of entity types that are defined in an Entity Data Model (EDM). |

* + 1. **Relations and their properties**

|  |  |
| --- | --- |
| **Connector** | **Properties** |
| **Allowed to use** | The layers are related to each other by the strictly ordered relation allowed to use. |
| **Uses** | The uses style shows how modules depend on each other; it is helpful for planning because it helps define subsets and increments of the system being developed. |

* 1. **Context diagram:**
  2. **Architecture background:**

System was separate into three layers include: Controller, Model, View

The reason why we use three layers instead of four layers (such as: Presentation layer, Business Logic layer, Data Object Transfer Layer, Data Access Layer) is:

* The controller in MVC undertake tasks that is implemented on Business Logic Layer and Data Access Layer. Moreover, View render a webpage to client as a UI and controller can support a view by returning appropriate view to client base on its request.
* On Web environment, increasing performance is important. Therefore, eliminating a layer is needed.
* Also, the scope of system isn’t large and doesn’t require much more complex business logic calculations, thus, MVC framework can fulfill its job.

Let’s get to the bottom of MVC framework:

* The view is responsible for providing the user interface (UI) to the user. It is given a reference to the model, and it transforms that model into a format ready to be presented to the user.
* The controller is responsible for responding to user input, often making changes to the model in response to user input. In this way, controllers in the MVC pattern are concerned with the flow of the application, working with data coming in, and providing data going out to the relevant view.
* The model that is used to send information to the Data Access Layer, perform business calculations, and even render in a view. Otherwise, these objects represent the domain of the application focuses on, and the models are the objects you want to save, create, update, and delete. In model, we use entity framework because its benefit:
  + With Entity Framework, the developers issue queries using LINQ, then retrieve and manipulate data as strongly typed objects. The Entity Framework’s ORM implementation provides services like change tracking, identity resolution, lazy loading, and query translation so that developers can focus on their application-specific business logic rather than the data access fundamentals.
  + Its goal is to decrease the amount of code and maintenance required for data-oriented applications. Entity Framework applications provide the following benefits:
    - Applications can work in terms of a more application-centric conceptual model, including types with inheritance, complex members, and relationships.
    - Applications are freed from hard-coded dependencies on a particular data engine or storage schema.
    - Mappings between the conceptual model and the storage-specific schema can change without changing the application code.
    - Developers can work with a consistent application object model that can be mapped to various storage schemas, possibly implemented in different database management systems.
    - Multiple conceptual models can be mapped to a single storage schema.
    - Language-integrated query (LINQ) support provides compile-time syntax validation for queries against a conceptual model.

There is the list of MVC# framework features that is the reason why we use this:

* *Views and controllers get connected automatically*. The MVC framework automatically establishes links between views and corresponding controllers. Therefore, the developers do not care about the associating views and controllers that linked to their views.
* *Multiple GUI platforms supported.* MVC allows targeting different GUI platforms such as: Window, Web, Silverlight, etc,… Therefore, the same application can be used with quite different presentation layers - one for Windows, the other for Silverlight or Web environment, etc.:



### Platform-independent navigation to views. To make application logic fully independent of the presentation layer, MVC provides a platform-independent way of navigating to views. Instead of  activating a Windows form or redirecting to a Web page a developer just simply call a uniform Navigator.Navigate(...) method. For example:

public class OrderDetailsController

...

public void ProcessOrder()

{

// No Response.Redirect(...) or Form.Show() calls

Task.Navigator.Navigate(OrderSupportTask.ProcessOrder);

}

### Tasks concept. Sometime, we have to unites several views with their controllers to do some job, this is called a task. For example a checkout task may consists of two views, one to choose a product (such as: Milk, Drink, Cake,… we can order in supermarket), the other – to do the payment. In MVC# all controllers within a task are given a link to the task object. Generally a task can be expressed as a workflow or a state machine.

### 

* 1. **Glossary of terms:**
  2. **Other information:**

1. **Data Model**



|  |  |  |  |
| --- | --- | --- | --- |
| Entity | Attributes | Data Type | Description |
| Bill | **Bill\_ID** | BIGINT |  |
| **POS\_ID** | VARCHAR(11) | Attribute said bill is made in which POS well as the general store |
| **Customer\_ID** | BIGINT | Customer pays the invoice. |
| **User\_ID** | INT | Cashier |
| TotalCost | FLOAT | The total cost of the bill, ensuring the implementation of Statistical Performance Data |
| Date | DATETIME | Paid Bill Date |
| PlusPoint | INT | Minus and plus points in a session will be stored here. |
| MinusPoint | INT |
| Product | **Product\_ID** | BIGINT |  |
| Product\_Name | NVARCHAR(50) |  |
| BasicCost | FLOAT | AttributeBasic Cost show the default price of the product |
| **Category\_ID** | VARCHAR(11) |  |
| Bill\_Detail | **Bill\_ID** | BIGINT |  |
| **Product\_ID** | BIGINT |  |
| Quantity | INT | Sum Loyal Point of customer |
| Customer | **Customer\_ID** | BIGINT |  |
| Customer\_Name | NVARCHAR(50) |  |
| Customer\_Address | NVARCHAR(50) |  |
| Customer\_Phone | VARCHAR(15) |  |
| SumPoint | INT | Sum Loyal Point of customer |
| RetailStore | **RetailStore\_ID** | INT |  |
| RetailStore\_Name | NVARCHAR(50) |  |
| Cost | **Product\_ID** | BIGINT | Entity Cost said that Retail Store Retail Store selling a certain product and pricing individual products within a certain time. |
| **RetailStore\_ID** | INT |
| DateStart | DATETIME |
| DateEnd | DATETIME |
| Cost | FLOAT |
| Category | **Category\_ID** | BIGINT |  |
| Category\_Name | NVARCHAR(50) |  |
| RetailStore\_Category | **RetailStore\_ID** | INT |  |
| **Category\_ID** | BIGINT |  |
|  | Quantity | INT |  |
| Computer | **Computer\_MAC** | VARCHAR(11) |  |
| **RetailStore\_ID** | INT | This Attribute tells us this POS Terminal is placed at which Retail Store |
| User | **User\_ID** | INT |  |
| User\_Name | NVARCHAR(50) |  |
| User\_Address | NVARCHAR(50) |  |
| User\_Phone | VARCHAR(15) |  |
| Password | VARCHAR(32) |  |
| **RetailStore\_ID** | INT | This Attribute tells us this user works at which Retail Store |

1. **Dynamic Perspective**

**Component and Connector View**

* 1. **Primary presentation:**



* 1. **Element catalog:**
     1. **Elements and their properties**

|  |  |  |
| --- | --- | --- |
| Elements | | Properties |
| Client tier | **Web Browser** | A component that send a request to web server and receive a reply is sent by web server. And then it displays UI to user. |
| Server Tier | **IIS 7** | IIS 7 is a major enhancement to the Windows web platform and plays a central role in unifying Microsoft web platform technologies - ASP.NET, Windows Communication Foundation web services, and Windows SharePoint Services.  It handle requests and replies that are send between client tier and server tier. |
| **ProductControllers** | The ProductController class contains action methods that render view pages (AddProduct, EditProduct, ViewProduct) |
| **Category**  **Controllers** | The CategoryController class contains action methods that render view pages (AddCategory, EditCategory, ViewCategory) |
| **StatisticsControllers** | The StatisticsController class contains action methods that render view pages (Statistics) |
| **SaleControllers** | The SaleController class contains action methods that render view pages (BillManagement, Checkout, PriceLog) |
| **StoreControllers** | The StoreController class contains action methods that render view pages (AddStore, ViewStore, EditStore) |
| **LoyalMember**  **Controllers** | The LoyalMemberController class contains action methods that render view pages (AddLoyalMember, ViewLoyalMember, EditLoyalMember) |
| **StoreCategory**  **Controllers** | The StoreCategoryController class contains action methods that render view pages (AddStoreCategory, ViewStoreCategory, EditStoreCategory) |
| **ComputerControllers** | The POSController class contains action methods that render view pages (AddPOS, ViewPOS, EditPOS) |
| **UserControllers** | The UserController class contains action methods that render view pages (AddUser, ViewUser, EditUser) |
| **AddStore** | This GUI helps the user to add a new Store. |
| **ViewStore** | This GUI helps the user to view the Store. |
| **EditStore** | This GUI helps the user to modify some information about the Store. |
| **AddCategory** | This GUI helps the user to add a new Product Category. |
| **ViewCategory** | This GUI helps the user to view the Product Category. |
| **EditCategory** | This GUI helps the user to modify some information about the Product Category. |
| **AddProduct** | This GUI helps the user to add a new Product. |
| **ViewProduct** | This GUI helps the user to view the Product. |
| **EditProduct** | This GUI helps the user to modify some information about the Product. |
| **BillManagement** | This GUI helps the user to view the list of Bills and the user can see some detail information of each Bill. |
| **PriceLog** | This GUI shows all Price history that was used for each product |
| **Checkout** | This GUI helps the user check bill and make a payment. |
| **AddUser** | This GUI helps the user to add a new User. |
| **ViewUser** | This GUI helps the user to view the User. |
| **EditUser** | This GUI helps user to modify some information about the User. |
| **AddLoyalMember** | This GUI helps the user to add a new loyal Customer. |
| **ViewLoyalMember** | This GUI helps the user to view the loyal Customer. |
| **EditLoyalMember** | This GUI helps user to modify some information about the loyal Customer. |
| **AddStoreCategory** | This GUI helps the user to add a new Retail Store Category. |
| **ViewStoreCategory** | This GUI helps the user to view the Retail Store Category. |
| **EditStoreCategory** | This GUI helps user to modify some information about the Retail Store Category. |
| **AddComputer** | This GUI helps the user to add a new Computer. |
| **EditComputer** | This GUI helps user to modify some information about the Computer. |
| **ViewComputer** | This GUI helps the user to view the Computer. |
| **Statistics** | To make statistics about the total amount of product (or product category) was bought on month. |
| **ProductModels** | Storing and retrieving the Product information and return a message back to view pages (AddProduct, EditProduct, ViewProduct) |
| **SaleModels** | Storing and retrieving the Sale information and return a message back to view pages (BillManagement, Checkout, PriceLog) |
| **StoreModels** | Storing and retrieving the Retail Store information and return a message back to view pages (AddStore, EditStore, ViewStore) |
| **LoyalMember**  **Models** | Storing and retrieving the Customer information and return a message back to view pages (AddLoyalMember, EditLoyalMember, ViewLoyalMember) |
| **UserModels** | Storing and retrieving the Userinformation and return a message back to view pages (AddUser, EditUser, ViewUser) |
| **CategoryModels** | Storing and retrieving the Product Categoryinformation and return a message back to view pages (AddProductCategory, EditProductCategory, ViewProductCategory) |
| **StoreCategory**  **Models** | Storing and retrieving the Store Category information and return a message back to view pages (AddStoreCategory, EditStoreCategory, ViewStoreCategory) |
| **ComputerModels** | Storing and retrieving the POSinformation and return a message back to view pages (AddPOS, EditPOS, ViewPOS) |
| **StatisticsModels** | Storing and retrieving the some information related Statistics and return a message back to view pages (Statistics) |
| **TempDB.xml** | When client send a request to get data from DB to display UI. The system will store this data in TempDB as a temporary data. Afterwards, if client need this data, and sent request to Data Access, it will read TempDB file and return data to client to display UI, needless to access database. |
| Database tier | **Primary Database** | It’s main database server which uses frequently when the system work properly. It’s responsible for store data such as sales data, user data, customer data, store data, product data, and category data. In the certain time, it will synch with temporary database as a backup data. |
| **Backup Database** | It’s a temporary database which uses rarely. It only uses when the system doesn’t work properly, crash or not available. In the certain time, it will synch all data with primary. |

* + 1. **Relations and their properties**

|  |  |
| --- | --- |
| **Connector** | **Properties** |
| **Request/ Reply** | Connector between client and server style, used by a client to invoke services on a server. |
| **Call and return** | Responsible for conveying the service request from the requester to the provider and for returning any results. Use by interface to request data from Filter/Object |
| **Synchronous Replication** | The technique for replicating data by two or more databases (or file systems) where the system being replicated does waits for the data to have been recorded on the duplicate system before proceeding. |
| **File I/O** | Refer the communication in access data in a file to perform operation such as: read file and write file. |
| **ODBC** | Stands for Open Database Connectivity. It is the standard method which allows any application to connect data. ODBC uses a middle layer called the database driver to handles the connection in between the application and the relational database management system. |

* + 1. **Element behavior**
       1. **Add Store**



* + - 1. **Add Loyal Member**



* + - 1. **AddUser**



* + - 1. **AddUserComputer**



* + - 1. **Statistic**



* + - 1. **AddNewCategogy**



* + - 1. **AddNewProduct**



* + - 1. **Sale**



* 1. **Architecture background:**
     1. **Rationale design**

The system includes 2 Database Server located in Head Office and Store and application uses data from these servers for sales activities as well as data storage. So in this architecture will use the Call-Return Styles, include Client-Server style and a Call-return style that objects call other objects referred to other data and wait for the return data from them. The called object will be call data from repository use Client-server styles. They will send request to database server and wait for the reply from them.

* + 1. **Analysis of results**

When use Client-Server style with 2 database server located at 2 different places we can improve system availabilities

* + 1. **Assumptions reflected in the design**

There will have reserve database server in store, which is responsible for storing product information daily and sales information to sync up to Head Office server and performing the redundancy while Head Office server going down.

* 1. **Glossary of terms:**

**DB Server:** Database Server is a computer program that provides database services to other computer programs or computers.

**POS Terminal:** A point-of-sale terminal is a computerized replacement for a cash register.

1. **Solution background**
   1. **Architectural Approaches**
      1. **Ensuring Performance:**

The system must handle requests quickly even when there are many requests

Introducing concurrency: Load balance can use to balance the requirements for the two servers

Increase resources available: use two servers to be able to process multiple requests

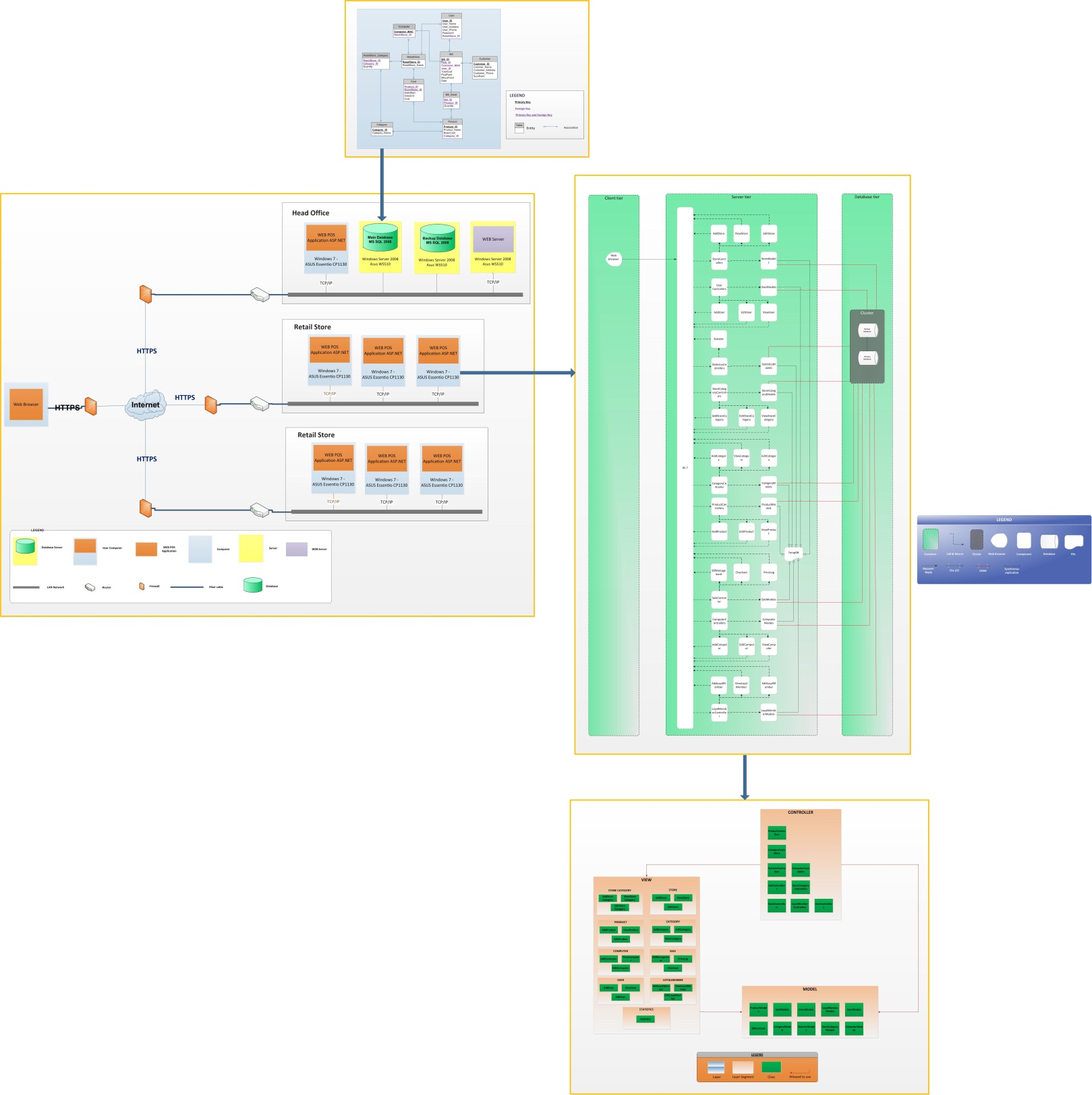
Maintain multiple copies of either data or computations: When scanning a product code of the product, the system will get the product information and store it on a cache, in the second times scan that product, the product information will be retrieved from the cache

* + 1. **Ensuring Performance:**

**Ensuring operation in POS that should be performed rapidly.**

* Reduce computational overhead: To keep it is quickly and timely in performing operation, the system will reduce a communication between tiers with each other. However, the mobility of system will be decreased.
* Increase available resources: increase hardware (real-time processing, memory, transmission...). However, the cost will be increased.

1. **Mapping between perspectives**



* 1. **Mapping between module view and allocation view.**



|  |  |
| --- | --- |
| Element in Allocation View | Element in Module View (Data Model) |
| Main Database | **Head Office:**   * + Bill   + Bill\_Detail   + User   + Product   + Category   + Cost   + POS   + Customer   + RetailStore\_Category |

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

## -- The End --