



Abstract

This document proposes a design for modernizing the application intake process, underwriting process, policy issue and policy maintenance process.

Version 1.1

**Electronic Insurance Policy**

High Level Design Document

Contents

[1 Introduction 3](#_Toc482009604)

[2 System Overview 4](#_Toc482009605)

[3 Design Considerations 6](#_Toc482009606)

[3.1 Assumptions and Dependencies 6](#_Toc482009608)

[3.2 General Constraints/Limits 6](#_Toc482009610)

[3.3 Goals and Guidelines 6](#_Toc482009611)

[4 NFR Considerations 7](#_Toc482009612)

[4.1 Usability 7](#_Toc482009613)

[4.2 Reliability 7](#_Toc482009614)

[4.3 Performance 8](#_Toc482009615)

[4.4 Security 9](#_Toc482009616)

[4.5 Supportability 9](#_Toc482009617)

[4.6 Infrastructure Requirements 10](#_Toc482009618)

[4.7 Implementation Constraints 10](#_Toc482009619)

[5 Solution Architecture 12](#_Toc482009620)

[5.1 Components Description 12](#_Toc482009621)

[6 Framework/Infrastructural Components 14](#_Toc482009622)

[6.1 Logging 14](#_Toc482009624)

[6.2 Exception Handling 14](#_Toc482009625)

[6.3 Security 14](#_Toc482009626)

[6.4 Configuration 14](#_Toc482009627)

[6.5 Caching 14](#_Toc482009628)

[6.6 Alert & Notification 14](#_Toc482009629)

[7 Integration (External Systems) Architecture 15](#_Toc482009630)

[7.1 Solution Integration Diagram 15](#_Toc482009631)

[7.2 Integration Components Description 15](#_Toc482009632)

[8 Deployment Architecture 17](#_Toc482009639)

[8.1 Deployment Diagram 17](#_Toc482009640)

[8.2 Components Description 17](#_Toc482009641)

[8.2.1 Presentation Layer 17](#_Toc482009644)

[8.2.2 Service Layer 17](#_Toc482009645)

[8.2.3 Persistence Layer 18](#_Toc482009646)

[8.2.4 Database Layer 18](#_Toc482009647)

[8.2.5 Third Party Services 18](#_Toc482009648)

[8.2.6 Scheduler Service 18](#_Toc482009649)

[8.2.7 Document Repository 18](#_Toc482009650)

[9 Solution Maintenance Tips and Caveats 19](#_Toc482009651)

[9.1 What to do to add a new module? 19](#_Toc482009652)

[9.2 What to do to run application in debug mode? 19](#_Toc482009653)

[9.3 …… 19](#_Toc482009654)

[10 Glossary 20](#_Toc482009655)

[10.1 Requirements Traceability Matrix 20](#_Toc482009656)

[10.2 Terms 20](#_Toc482009657)

[10.3 Technology POCs 21](#_Toc482009658)

[10.3.1 <POC1> 21](#_Toc482009659)

[10.3.2 <POC2> 21](#_Toc482009660)

# Introduction

“YCompany” is one of the renowned name in Insurance providers in industry, providing term and life insurance to customers in US. The company's product offerings include term and life insurance.

The company is planning to modernize the application intake process, underwriting process, policy issue and policy maintenance process.

This document describes the current process in place and System architecture & high level design of the new solution being proposed.

The specifications described in this document can be used to initiate development of the application. However, the design described would require appropriate approvals from the company.

# System Overview

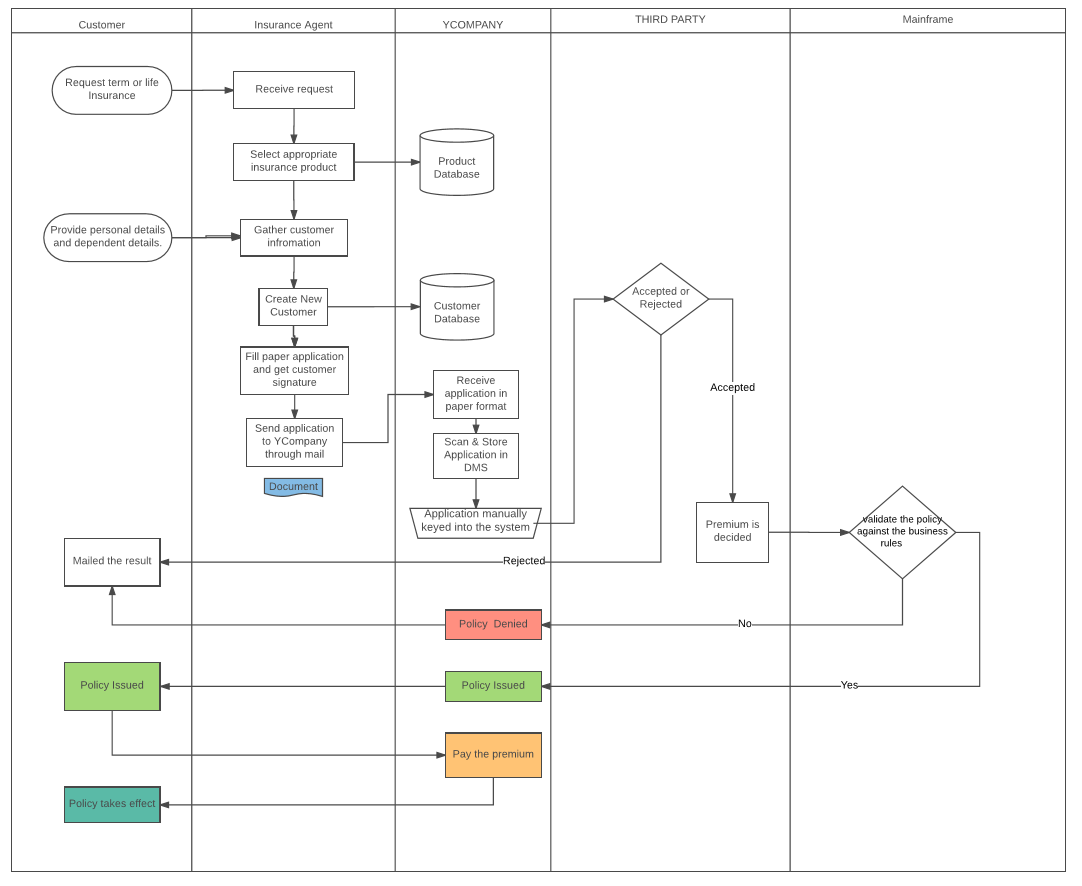
YCompany serves more than 200 million customers across various geographies. Being a key player in its business area they have been facing challenges with the manual process currently in place.

Hence, the company is planning to modernize its existing process by building different portals for different processes currently in place. They have collaborated with 3rd party to provide underwriting service or them and they expose service to be consumed for any application coming it. The main expectations from these portals are mentioned below:

* Customer Self Service Portal
  + Customer can **create a login using their policy details**
  + Customer should be able to change the correspondence address
  + Customer should be able to change the billing cycle
  + Customer should be able to make electronic payment against policy premium.
* Agent Portal
  + Agents should be able **to login to the Agent Portal**. The login for Agents are created when they register with the company for the first time.
  + Agent should be able to submit the policy electronically.
  + Agent should be able to get eSignature of the customer
  + Agent should be able to look at the current status of the application.
  + Agent should be able to see report of the applications submitted and converted to policy
  + Agent should be able to see the total premium for the policy issued by them.
* Internal Portal (Underwriter/Admin/ Billing Department /Auditor)
  + Internal user should be able **to login to the portal.**
  + Internal user should be able to look at the incoming application.
  + Internal user should be able to update incoming application if any rectifications are required.
  + Internal user should be able to look at the underwriting results coming from a third party and should be able to edit or update the underwriting result if required.
  + Underwriting data should be stored for further auditing.
  + Internal User should be able to generate reports on the total volume of application coming in and conversion to policy.
  + Internal User should be able to generate report for policy defaulting on payment.
* Process Automation
  + Intake process of application to send data to mainframe and send policy data to 3rd Party for Underwriting.
  + Receive underwriting results and post received data is automatically processed in mainframe in the batch cycle.
  + Any rejects from mainframe cycle should be notified.
  + Integration with 3rd party payment provider should be done for premium collection.

Summarizing the process flow as follows:

**Process flow chart:**



# Design Considerations

## Assumptions and Dependencies

* Proper functionality of system would be dependent on the performance of 3rd party services being used in the system like mainframe, underwriting & payment provider.
* 3rd party to provide underwriting service for the company and the said exposed service to be consumed for any application consuming it.
* Internal 3rd party reporting services will be used.
* Consistent access to mainframe for initial verification of application information.
* Consistent & Licensed access to payment providers.

## General Constraints/Limits

Proposed solution would be making easier for the company to manage existing manual process for issuing policies. Only authorized personnel will be allowed to access the system. The system will have users of type agent, administrator, internal user, biller, underwriter and policyholder. The infrastructure required for the system would be Windows Server 2012, SQL Server 2012 and IIS Server to host the application. Application would be compatible with most running browsers today but it would work most efficiently with Internet Explorer and Google Chrome.

## Goals and Guidelines

The proposed design for the application will be based on the following design & architectural principles

* Design for evolution, flexibility and reuse by dividing the project in layers. Layers are logical separation within the code. This helps us in separating concerns. Logical layer supports testing. Properly separated logical layers support automated testing by defining interfaces through which these layers interact. Layers also minimizes dependencies with the application making it loosely coupled all through. Minimizing dependencies makes code easier to understand and maintain.
* Functionality componentize as services.
* Environment would be up and running 24 x 7. In case of any crashes system should again restart on its own.
* System should be reliable.
* System should have enough logging to help in debug any error condition.
* Services as Web APIs will be implemented for interaction with UI layer,
* All services will follow RAML standards.
* Angular2 with Typescript will be used for presentation layer and will consume data from the web services.
* Token based authentication to be used so that the same mechanism can be used for web and mobile application.
* Automated build with continuous integration.
* Continuous Code quality to be setup with CI for any code vulnerabilities and bugs found during build time.

# NFR Considerations

## Usability

* Policy Holders, Agents, internal users, admin users and other users based on their role will be able to login to the application.
* All kinds of input validation will be part of applications business logic.
* Application will be developed in English.
* Application is developed in line with the current process to keep the learning curve to a minimum for the existing users. Detailed user manual will be constructed*.*
* For documentation a *Chm help file can be provided containing sections for each type of user.*

## Reliability

* Number of users logged in at a time can be managed and restricted to handle & optimize the time required to deliver a request.
* Ease of Recovery
* How should the system respond to errors?
* *Error Handling library will be responsible to take care of errors and provide back up mechanisms in scenarios of system recovery.*
* Maximum Known Bugs

Application will not be deployed with any high severity bugs or 2 to 5 medium severity bugs.

* High severity would be loss of Business Use case/ functionality or performance related
* Medium severity would be in Reports or UI.

## Performance

* Throughput
* How many concurrent users may be using the system at any given peak-use time?
  + It will be calculate based on system resources and request execution time.
* How many additional resource requirements (server, load balancers, etc.) are acceptable by the system to maintain acceptable latency and throughput with increasing load?
  + Based on the above calculation we can decide.
* What is the acceptable request range that must be catered to in a given amount of time under a given operating load?
  + Also depends on the above calculation.
* Response Time
* What is the acceptable response time range (in seconds) to a user action on the system?
  + < 1sec.
* Resource Usage
* Are there size or capacity constraints on the data to be processed by the system?
  + Yes and system upgradation would be required on time to time basis.
* Are there any CPU usage constraints while the application is executing?
  + CPU standards should be met , dual core, i7, 8GB RAM etc.
* Degradation Under Overload Conditions
* How should the system respond to extreme conditions?
  + Please refer Infrastructure diagram section.

## Security

* Internal Security
* Must access to any data or the system itself be controlled?
  + Data security mechanism is implemented to avoid anti-cross site scripting and sql injection.
* Are there any requirements to ensure the integrity of the system from accidental or malicious damage?
  + At data level, ACID properties will be maintained.
* External Security
* Must all external communications between the system’s data server and clients be encrypted?
* Encryption mechanism is implemented at each request level.

## Supportability

* Ease of Installation
* Is there any need for an automatic installation package? Or manual installation will suffice? NA
* Who is responsible for system installation? System Admin.
* How many installations of the system will be required? 1 per site.
* Planned Maintenance
* What would be the frequency of system maintenance? NA
* Who will be responsible for system maintenance? System Admin.
* Modification
* What parts of the system are likely candidates for later modification?
* What sorts of modifications are expected?
* Backup
* How often will the system be backed up? Scheduler is implemented as per requirement. (Weekly or daily)
* Who will be responsible for the back up? Scheduler.
* Ease of Configuration
* What type of configuration requirements of the system?
  + Service timeout changes, multi lingual etc.
* Is any type of administration panel is required to configure the system? System admin can login and configure the master data.
* Would configuration items specified in application configuration file be acceptable? Yes.
* Ease of Testing
* Is system needs to be testable via automated test cases? Yes
* What level of user acceptance testing is required? NA
* Are there any requirements to load test the system? Yes
* What level of feature isolation/factoring is required from testability perspective?
  + All Features will be tested in an isolated way through Unit testing.

## Infrastructure Requirements

* Clients
* What are the client system hardware, memory and browser requirements?
* compatible browser would be required.
* Servers
* What hardware is the proposed server system to be used on?10GB of RAM, 1TB of Hard disk. Windows Server 2000 and above.
* Networks
* What is the connection bandwidth required connecting Web Server and Application Server? On lower bandwidth performance will be affected however it will work.
* Peripherals
* Any specific hardware devices required running the system? NA
* Web Services
* What are the external web services on which system will depend on?

## Implementation Constraints

* Development Platform
* Is there any specific requirement that the system should be implemented on a particular platform such as .NET 3.5?
  + Reusing existing components and reducing time to market.
* Languages
* Is there any specific requirement that system should be implemented in a particular programming language?
* OOPs is preferred.
* Operating Systems
* Is it important that the system be portable (able to move to different hardware or operating system environments)?
* Yes it can be used on mobile handsets also
* What Operating systems application should be able to execute on?
* Windows.
* Standards
* Are there any specific implementation standards that need to be followed during development?
* Component based programming, testing framework , optimization tools, coding guidelines. Component review tools.
* System Interfaces
* How many different types of interfaces (presentation layers) required for the system?
* Minimum web interface is located.
* Is input coming from systems outside the proposed system?
* No
* Is output going to systems outside the proposed system?
* Yes
* Are there restrictions on the format or medium that must be used for input or output?
* Yes.
* Legacy Systems
* Is there any type of interaction required with any existing legacy system?
* Yes
* Databases
* What database implementation application needs to support?
* SQL server
* Is support for more than one database required?
* Yes both server, sonarqube and mainframe
* Will there be any database partitioning or mirroring required?
* NA

# Solution Architecture

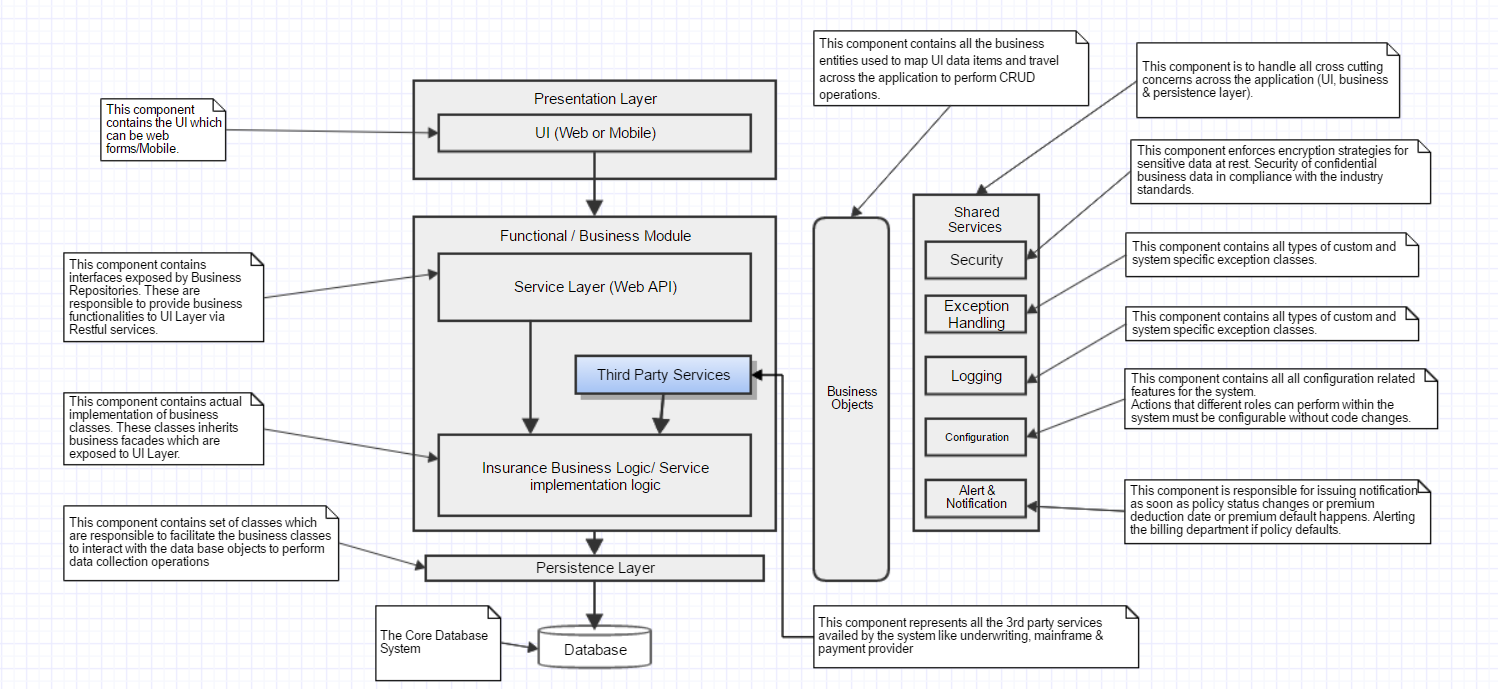


Figure 1: Solution Architecture Diagram prepared in Gliffy.

## Components Description

### Presentation Layer

This layer represents the UI of the application. It contains the user related functionality for managing interaction with the application. It will make service calls in the form of http requests to the Web API layer which then communicates with the business logic layer. In this project, it can be a web application or a mobile application. Keeping this layer helps in abstracting away the complexity by pushing most of the code to other layers. Its core responsibilities would be:

* Provide graphical interface to end users.
* Validation of input data.
* Outputs the processed data.
* Calls functional/business facades to process and save data.

### Service Layer (Web API)

This layer separates the presentation layer from the actual business logic layer. It will handle interaction between UI and complex interaction with the business layer. In this application, this is going to be Restful Web API. Presentation layer will make http requests to this layer over the network. This layer will then take requests and fire them off to business layer, but adding value by determining which request should be called in what order, and managing the complex interactions. This layer will delegate work to business objects. It follows a transaction script pattern i.e. a procedural set of steps to fire off different methods within the business layer. It will not contain any logic to truly perform the work.

This will also take care of all the cross cutting concerns i.e. Security implementation like checking authorization, and user roles would be part of this layer, handling alert and notification to listeners, any data validation making sure that the data received from the UI layer meets the requirements for a given context, logging, configuration & error handling.

This layer would use IOC container for Dependency Injection. In the POC project, I have used Unity Framework by Microsoft to inject dependencies across different API controllers. There are few other IOC containers like Ninject also available.

### Business Layer

The business functionalities are logically grouped under this layer. This layer consists of the service contracts to offer the business logic to separate the business layer as an independent layer. It is an implementation of the Facade pattern. These facades are exposed to facilitate the client/ UI layer to access the concrete functionalities provided by Business Repositories.This layer stands alone and can be reused elsewhere. It also implements the core functionality of the system and encapsulates the relevant business logic. This layer avoids the presentation layer depending directly on our database. It manages the behaviors that trigger from the presentation layer, and ultimately interacts with the data access layer. The business repositories are mapped and registered with business facades using dependency injection.

### Third Party Services

This is a set of interfaces that would help in integrating with domain specific third party service providers, like underwriting service, access to mainframe, payment provider & document repository. Technically, these are combination of abstract components which can be extended by introducing new concrete components to integrate with any updated versions of third party service providers.

Mainframe will be used for verification of the customer.

Underwriting service to validate and provide a face value for the policy

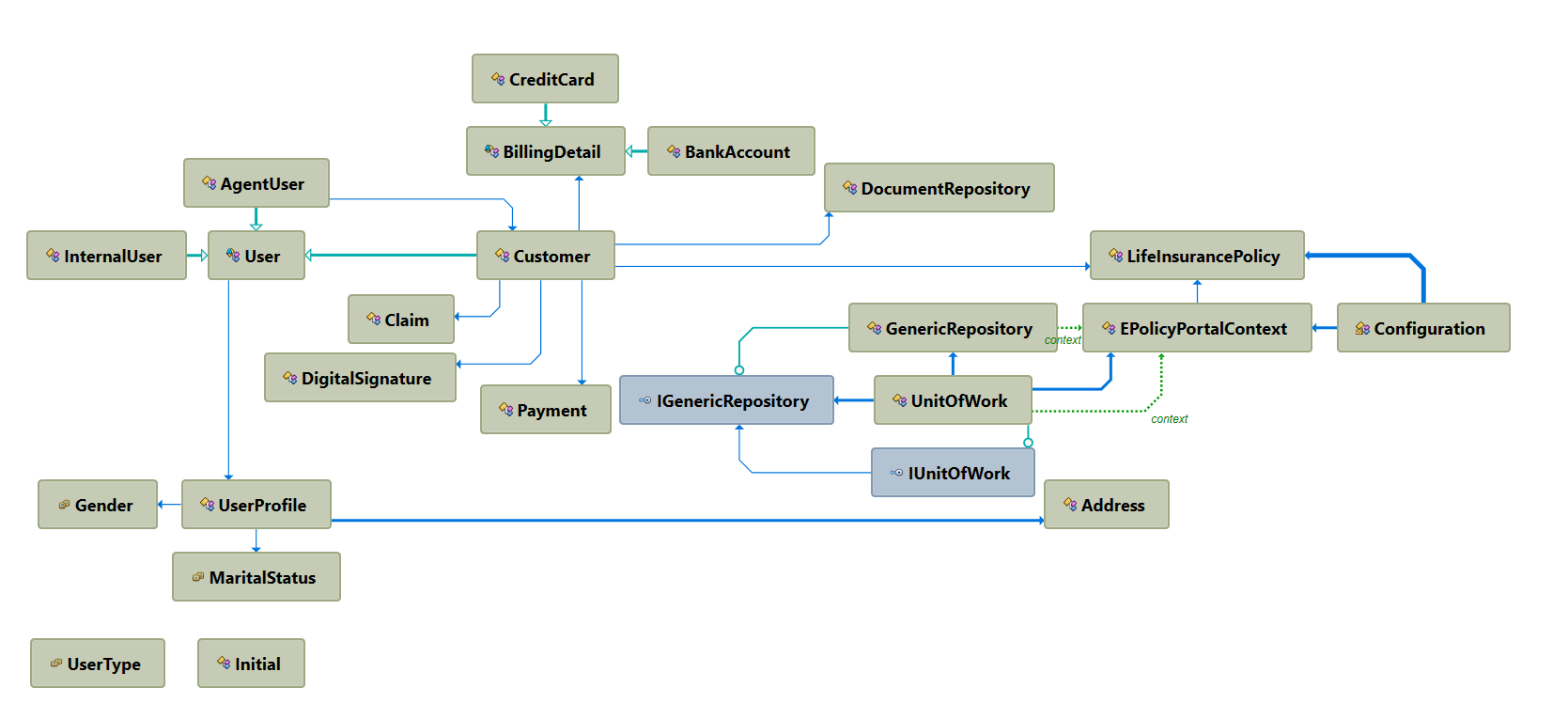
Payment Provider to provide the payment infrastructure for the application.

Document Repository to store the document proofs of the policy holders. It the place where all the scanned verified documents of the customer would be stored.

### Persistence Layer

This layer communicates with database to retrieve and save the data in database using its own context. This data access layer exposes generic interfaces that the components in the business layer can consume. It uses an ORM to interact with the database. It implements Repository & Unit of work patterns to manage transactions. The application uses code first approach to construct the database during deployment as well. All the master data of the application will be part of code-migrations Seed method.

Below a Type Dependency diagram in this layer.



### Database

This layer contains the actual business raw data. The database would include the following tables:

1. **Customer**: contains, biographical, dependents and payment information.
2. **LifeInsurancePolicy**: contains information related to all the policy products available with the company.
3. **User**: contains information related to login details of a user.
4. **BankAccount**: contains information about the bank account details for payments of premium.
5. **CreditCard**: contains information about the credit card number of the customers. A customer can have more than credit cards information associated with it.
6. **Payments:** contains information about the payments, their due date etc. of the customers.
7. **Claims**: contains information about the claim settlements done and amount disbursed.
8. **Person** table is the root table to store basic details, contact information etc. This will be common for both the internal users and customers.
9. **Address** table will contain the information about addresses.



Figure 2: ER Diagram

### Scheduler Service

This will be the service responsible for generating appropriate alerts, feeds and notifications both for the customer and billing department for any customer defaults. Application would using SQL server Agent to send automatic notification through emails to the customers & billing department.

### Reporting Services

A separate dashboard would be provided for reporting which will present several types of reports for different types of users. They will be configurable using the configuration module. For this SQL Server Reporting Services would be used which is a solution that customers deploy on their own premises for creating, publishing, and managing reports, then delivering them to the right users in different ways, whether that’s viewing them in web browser, on their mobile device, or as an email in their in-box.

# Framework/Infrastructural Components



## Logging

Log4Net will be used as logging framework for generating log files. Different log files will be generated from the system. Server logs, client logs, module logs etc.

## Exception Handling

Custom exception handling will be present with error reporting features to collect details about the system when the exception occurred that may be helpful in fixing the problem.

## Security

Application will utilize an identity management system with role-based user authentication and authorization for allowed secured access. Application will be available through https link.

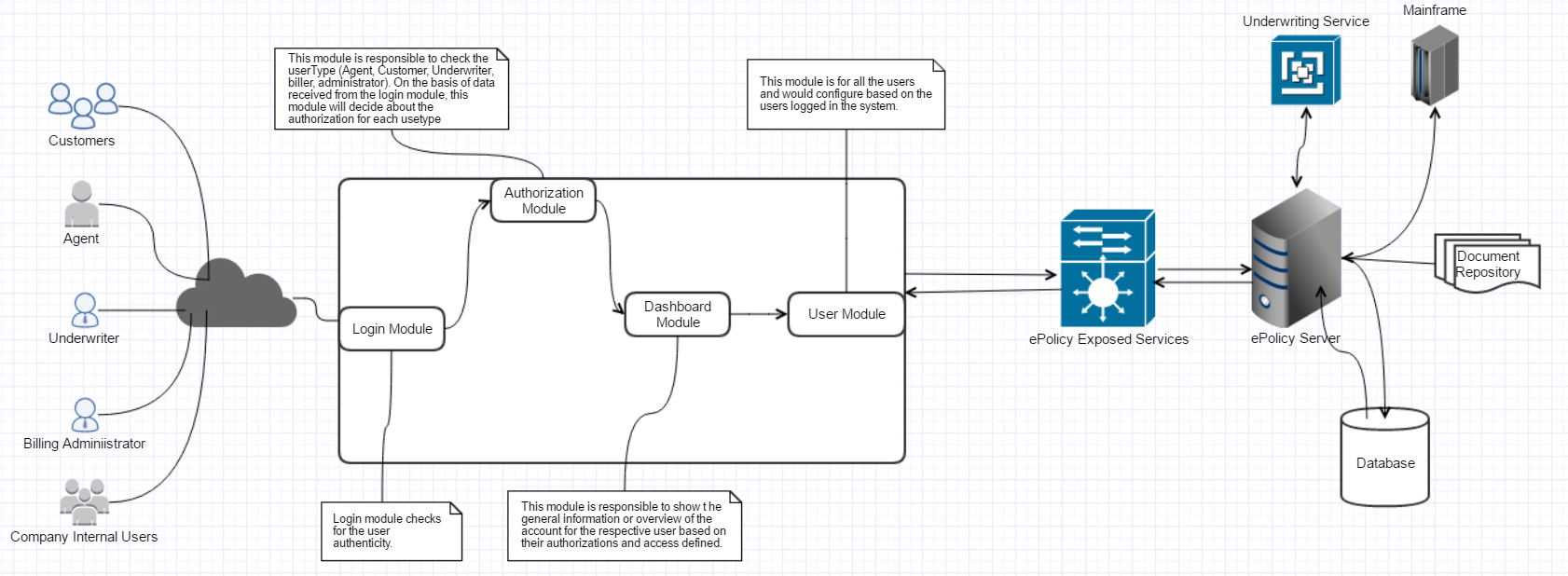
## Configuration

Application features would be configurable using this module. This will essentially read a config file which would be available in the installation directory. By changing settings in this file, admin would be configuring the application for different customers.

## Alert & Notification

Application would be notifying the users about various stages through this module. This module will take care of any notifications for the users for due payments and the billing department for any defaulters. It will be dependent on the input from the scheduler service.

# Integration (External Systems) Architecture



## Solution Integration Diagram

The purpose of this diagram is to explain how the different components (internal & external) are interacting with each other.

## Integration Components Description

The main entities of the proposed system would be e-Policy Portal application, e-Policy Exposed services & all the domain specific third party services like underwriting, mainframe, payment provider & document repository access.

### e-Policy Portal application

This application would be responsible for providing the user interface to interact with the system. Using this application user will be able to perform all the policy related activities like viewing policy details, issuing policy, accepting or rejecting policy or claims, make payments, generate reports, etc. The UI will interact with the e-Policy server application through exposed services.

### e-Policy Exposed services

This application will expose services to interact with the e-policy application. This will expose relevant restful api which can be accessed over the internet. This will work on claim based authentication.

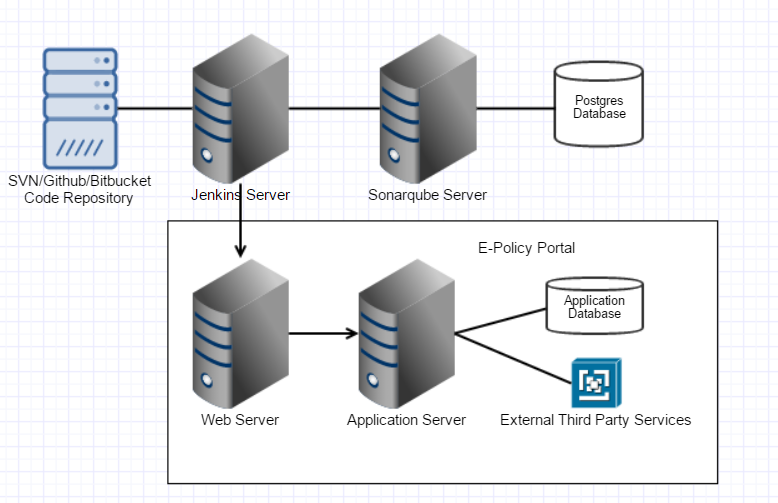
### e-Policy Application Server

This is the application server which will contain the actual business implementation and logic of the application. It is the main system which will provide the main functionality of the insurance system.

# Deployment Architecture

It represents how the complete system will be deployed in the production.

## Deployment Diagram



## Components Description

### Code Repository

Central code repository to maintain the application source. code base. These are important in a distributed development environment for version control. Developers will push and pull code into the repository. For this application, we can use Github. Some of the key features of Github include Integration with several applications and tools, integrated issue tracking. Every GitHub repository is also a Subversion repository.

### Jenkins Continuous Integration Server

Jenkins is an open source Continuous integration server. It is helpful in providing continuous integration and continuous delivery of nightly builds for the packaged application. One can create different pipelines for different branches in it. It will be configured to pull the latest code from the GIT repository into the pipeline and trigger automatic builds.

### Sonarqube Continuous Code Quality Server

Sonarqube is also an open source tool for continuously analyzing and measuring source code quality. It can be integrated with continuous builds as a build step in Jenkins. It combines static and dynamic code analysis. It allows development teams to monitor project health by identifying bugs and code vulnerabilities. It also provides an effort estimate to resolve the bugs. It stores its reports in a database. Any database like MySQL, Postgre, SQL Server can be configured with Sonarqube server to store the analysis per build. Here we used Postgre as it is an open source database with greater scalability options.

### Web Server

Web server helps in request filtering, authentication, failed request tracing. Helps in implementing security and diagnostic features. It will be responsible for accepting HTTP requests from the clients and serving back HTTP responses like html webpages. IIS 7 Web server will be used for hosting the application

### E-Policy Insurance Portal Application Server

This is going to be another IIS hosting where e-policy portal application services will be deployed. Scripts deployed on web server will call application server to perform operations or return relevant data.

### Application Database

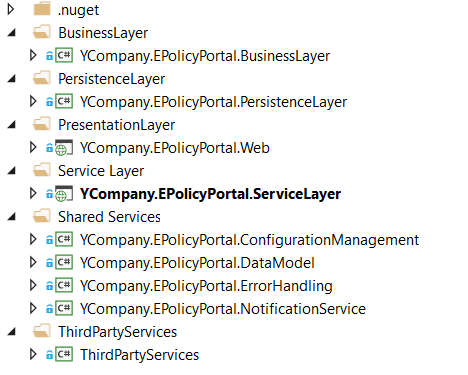
This database would be responsible for managing the application data. Sql Server will be used for the application database.

### External Third Party Services

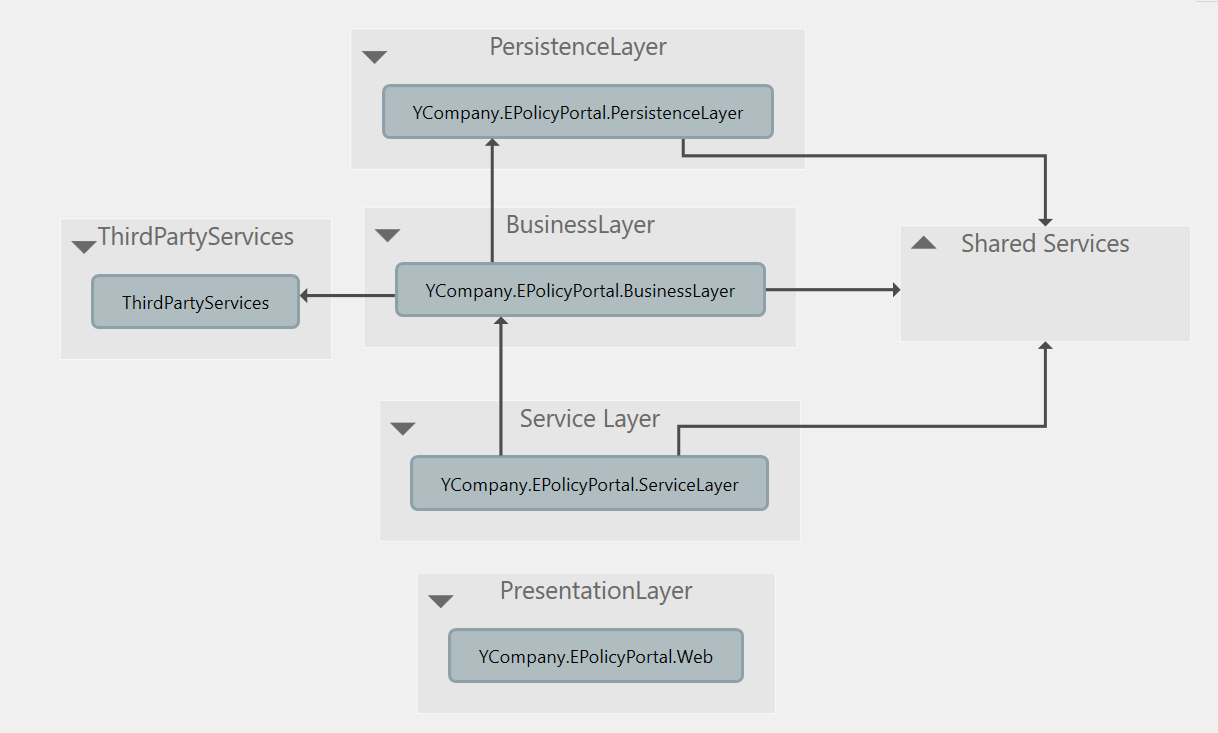
This component helps in consolidating all the domain services required for this application, like, underwriting service, payment provider service, mainframe & Document repository etc.

# Application Code Organization

Github will be used the repository



## Projects Dependency diagram

****

## Technology Stack

|  |  |  |
| --- | --- | --- |
|  | **Aspect** | **Recommendation** |
|  | Application Development | Microsoft Visual Studio 2015  Microsoft .NET 4.6  Microsoft ASP.Net Web API 2.0 – RAML Standards end points.  Angular 2.0 with Typescript – UI Layer |
|  | ORM | Entity Framework 6 |
|  | Database | SQL Server 2008 |
|  | Application Hosting | IIS Server 7.5+ with Windows Server |
|  | Reporting | Any third party reports for quick turnaround time like SSRS. |
|  | Continuous Integration | Jenkins can be used for continuous integration |
|  | Repository | Github |
|  | For continuous code quality | Sonarqube |
|  | Minification & bundling of scripts | Gulp |

# Solution Maintenance Tips and Caveats

## What to do to add a new module?

Add the new module as a class library in the solution under the relevant solution folder based on what type of module is being added the project. Add it to the Unity container instance in Unity.Config file which is under App\_Start folder in the Service Layer object. This is important so that

## What to do to run application in debug mode?

Service layer project should be start up project and run the application from visual studio.

# Glossary

## Requirements Traceability Matrix

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Epic ID** | **BR#** | **Module Name** | **UserType/ Role** | **Description** |
| EPIC-IPP-1 | IPP-1 | Login/Logout | Admin, Agent, Underwriter, Auditor, Biller, Customer | All the provided user types can login and logout from the application |
| EPIC-IPP-2 | IPP-2 | Add Customer | Agent | Insurance Agent can add a customer to the portal |
| EPIC-IPP-2 | IPP-3 | Get Customer | Admin, Agent, Underwriter, Auditor, Biller, Customer | All the provided user types can get a customer from the application. |
| EPIC-IPP-2 | IPP-4 | Update Customer | Admin, Agent, Underwriter, Auditor, Biller, Customer | All the provided user types can update a customer information in the application based on operation being performed. |
| EPIC-IPP-2 | IPP-5 | Deactivate Customer | Admin | Administrator can delete (soft delete) a customer in the application |
| EPIC-IPP-3 | IPP-6 | Add Insurance Policy | Admin | Administrator can add a new insurance policy to the application |
| EPIC-IPP-3 | IPP-7 | Get Insurance Policy | Admin, Agent, Underwriter, Auditor, Biller, Customer | All the provided user types can view an insurance policy details from the system |
| EPIC-IPP-3 | IPP-8 | Update Insurance Policy | Admin | Administrator can update policy details in the application. |
| EPIC-IPP-3 | IPP-9 | Delete Insurance Policy | Admin | Administrator can delete or deactivate policy details in the application. |
| EPIC-IPP-4 | IPP-10 | Pay Premium | Customer | Customer can login to the application and pay the policy premium |
| EPIC-IPP-5 | IPP-11 | Upload documents | Agent | Agent can upload the customer documents on the portal |
| EPIC-WMS-4 | IPP-12 | Issue Policy | Underwriter | Underwriter should be able to validate the policy details based on main frame results & issue the policy accordingly. |

## Glossary

Table 1: Glossary Terms and Definition

| Glossary Term | Definition |
| --- | --- |
| <Term one> | <Definition one> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam at porta est, et lobortis sem. Duis imperdiet in nisl sed luctus. Vivamus hendrerit molestie urna, eu viverra velit porttitor sit amet. Donec nec ligula vitae elit venenatis tincidunt. Curabitur in est placerat, lobortis ante sit amet, tincidunt tortor. Duis ligula dui, tincidunt ut aliquam quis, gravida in libero. Nunc vel lacinia turpis. Etiam metus neque, ultricies eget odio eu, mollis ultricies lectus. Vestibulum consectetur turpis eget diam dignissim posuere. |
| <Term two> | <Definition two> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam at porta est, et lobortis sem. Duis imperdiet in nisl sed luctus. Vivamus hendrerit molestie urna, eu viverra velit porttitor sit amet. Donec nec ligula vitae elit venenatis tincidunt. Curabitur in est placerat, lobortis ante sit amet, tincidunt tortor. Duis ligula dui, tincidunt ut aliquam quis, gravida in libero. Nunc vel lacinia turpis. Etiam metus neque, ultricies eget odio eu, mollis ultricies lectus. Vestibulum consectetur turpis eget diam dignissim posuere. |
| <Term N> | <Definition N> Lorem ipsum dolor sit amet, consectetur adipiscing elit. Etiam at porta est, et lobortis sem. Duis imperdiet in nisl sed luctus. Vivamus hendrerit molestie urna, eu viverra velit porttitor sit amet. Donec nec ligula vitae elit venenatis tincidunt. Curabitur in est placerat, lobortis ante sit amet, tincidunt tortor. Duis ligula dui, tincidunt ut aliquam quis, gravida in libero. Nunc vel lacinia turpis. Etiam metus neque, ultricies eget odio eu, mollis ultricies lectus. Vestibulum consectetur turpis eget diam dignissim posuere. |
|  |  |



**COPYRIGHT NOTICE**

Copyright © 2013-15 Nagarro Software Pvt. Ltd. **All rights reserved.**

This document is confidential and proprietary to Nagarro Software Pvt. Ltd. and no part of it should be reproduced, published, transmitted or distributed in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, or stored in any information storage or retrieval system of any nature nor should it be disclosed to third parties without the prior express written authorization of Nagarro Software Pvt. Ltd.

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History** | | |  |
| Version | Date | Author/Contributor | Comments |
| 0.1 | 10-04-2017 | Isha Gupta | First Draft |
| 1.0 | 18-05-2017 | Isha Gupta | Updated System specifications |