

**YCompany Electronic Claim Processing**

Design Document

[Abstract summary of the document]

Version 1.0.0

**Contents**

[1 Preface 2](#_Toc62565392)

[2 Introduction 3](#_Toc62565393)

[3 Overview 4](#_Toc62565394)

[4 Design considerations 7](#_Toc62565395)

[4.1 Assumptions & Dependencies 7](#_Toc62565396)

[4.2 Goals & Guidelines 8](#_Toc62565397)

[5 Non-Functional Requirement considerations 9](#_Toc62565398)

[5.1 Usability 9](#_Toc62565399)

[5.2 Reliability 9](#_Toc62565400)

[5.3 Performance 10](#_Toc62565401)

[5.4 Security 10](#_Toc62565402)

[5.5 Supportability 11](#_Toc62565403)

[5.6 Infrastructure Requirements 11](#_Toc62565404)

[6 Solution Architecture 12](#_Toc62565405)

[6.1 Solution diagrams 13](#_Toc62565406)

[6.1.1 Solution Design 14](#_Toc62565407)

[7 Framework/Infrastructural Components 20](#_Toc62565408)

[7.1 Logging 20](#_Toc62565409)

[7.2 Exception Handling 20](#_Toc62565410)

[7.3 Security 20](#_Toc62565411)

[7.4 Data Access 20](#_Toc62565412)

[7.5 Input Validation 20](#_Toc62565413)

[7.6 Object Creation 21](#_Toc62565414)

[7.7 Microservices communication 21](#_Toc62565415)

[8 Technology Stack 22](#_Toc62565416)

[9 Glossary 23](#_Toc62565417)

# Preface

This document will outline the software design and specification for proposed eClaims system for YCompany in addition to system architecture, system components, and software requirements as proposed upon by the project team.

# Introduction

YCompany decided to develop the Electronic Claim Processing application. This document describes the high-level design of the application. It explains design of the application by using solution diagram, detailed solution architecture, technology stack. It also covers non-functional requirements coverage, performance and scalability, assumptions, and scope. 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.

# Overview

“YCompany” is willing to build customer portal and mobile app which they can use to inform of an accident or loss, they should be able to submit any details regarding accident e.g., photograph of accident or police report and further details. Once the first information is received, the claims should be generated corresponding to the policy and incident. Incident manager, adjustor and surveyor should be notified of the incident. Customer should be constantly updated with the status of the claims processing.

Customer should be provided with a list of partner workshop where they can take the vehicle for repairs. Customer can choose from partner workshop or any external workshop. In case of partner workshop selection customer should be able to set up appointment with the workshop from the customer site itself. The partner workshop should be intimated of the initial accident details. Once the vehicle is dropped off based on the type of policy coverage user should be able to select a rental vehicle from the Car Rental partner from customer site itself. Any change in the status of the vehicle the customer should be notified.

Surveyor should be able to assess the damage on the vehicle once vehicle is in workshop and should be able to submit the report to back office, Once the assessment is submitted electronically, the adjustor should be able to evaluate the claim to be processed and approve the amount. Once it has been approved electronically Customer should be notified of the amount the claim has been approved for with notification also being sent to the workshop. The workshop should be able to keep updating the status of repairs which should be notified to the customer. The customer should be notified of the any changes in the expected delivery date of the vehicle based on information from workshop.

On completion of work the Customer should be able to pay the due amount electronically from the web site itself.

The partner workshop should be able to login to the site and upload work order with respect the car repairs and estimates. Partner Workshop should be able to track their payment based on the incidents they have addressed.

Claims department should be able to generate reports on claims processed, processing time, fraudulent claims, ageing matrix for claim if there are any claims which are pending from long time. Claims department should be able to crate claims processed reports based on different roles i.e., Adjustor, Surveyor, Case Manager, Workshop. All the information interchange should be electronic completely avoiding paperwork.

The claims received and processed should be stored in a central document management system for auditing and compliance perspective.

* Customer Portal and App
  + 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.
  + Customer should be able to submit claims.
  + Customer should be able to submit any supporting document.
  + Customer should be notified and alerted if any status change happens on claims.
  + Customer should be able to check the Partner Service providers (Repair workshop, authorized service station, Car Rental) based on this location or zip address.
  + Customer should be able to keep track of repair based on work order and progress report submitted by the Repair agency.
* Incident Management
  + Case Management team should be automatically assigned based on the availability of the Surveyor, Adjustor, Case Management also keeping in mind the Surveyor field office covers the area where vehicle is for repairs.
  + Surveyor should be able to submit their assessment online via website or app and once the surveyor assignment is submitted the adjustor should be notified.
  + Adjustor should be able to view the claim, supporting document, assessment report online and should be able to adjudicate the claim based on policy coverage.
  + In case Surveyor or Adjustor is not available for claims processing the case, manager should be able to delegate the claim processing to another Surveyor or Adjustor.
  + Case Manager should be able to view the complete details of the case and make adjustment to claims which can override the Adjustor and surveyor.
  + Auditor can look at all the details pertaining to the claims and any processing.
* 3rd Party integrations
  + 3rd Party service providers like in network Vehicle Repair Workshop should be able to provide detail work order and estimates with respect to claim.
  + They should be able to provide constant updates with respect to the work that is being done and has been completed.
  + Updates provided by them should be notified to customer (email/SMS) along with the delivery date.
  + They should be able to provide final bill amount for the repairs, which customer should be notified prior of and should be able to pay off from the site.
  + 3rd Party provider should be able to track the claims payment status.
* Internal Users Reporting
  + Internal user should be able to login to the portal.
  + Only Authorized Case Managers should be able to generate reports with respect to the claims they have received.
  + Regional Manager should be able to see the details with respect to claims received in their region with details like
    - Claims Processing Time
    - Claims amount paid out.
    - Number of claims with respect to geography.
  + Top management should be able to generate reports based on how each region is processing with respect to claims, which region more claims are being received, which region has higher claims amount paid out.
* Document Management
  + Document Management is the key aspect as every claims document should be saved and archived for further auditing and processing.
* Alerts and Notification
  + Alerts and Notification should be sent out constantly to all the relevant parties notifying them of any progress or status change of claims.
  + All communication with the customer should be archived for future reference, auditing, and compliance.

# Design considerations

## Assumptions & Dependencies

* Solution is only considering Auto loan, other insurance like General, Health etc. are out of scope of this design.
* Car workshop and rental providers already have the software system ready and we will be integrating it with e-claim system.
* YCompany will provide one time data entry team to backfill existing policies details via excel in defined format, this backfill will cover all existing policies to be inserted/uploaded on database server.
* Majority percentage of the user do have access to smart phones and internet access and well aware of using any website, though user training will be conducted before the application launch.
* Best suited browser is Google Chrome.
* Client has accepted cloud-based architecture, currently there is a discussion pending whether it will an on-premises or privately owned environment.
* YCompany Cloud partner is Microsoft Azure.
* Data centers will be cloud region based so as to follow security guidelines as per region/zone jurisdiction rules about data privacy.

## Goals & Guidelines

* System should be:
  + Readable
  + Correct
  + Reliable
  + Reusable
  + Extendable
  + Flexible
  + Efficient
* Iterative Development – As to resolve software flaws or risks before there has been a lot of time and effort put into the software, this approach enables continuous testing and continuous integration which creates the opportunity for early feedback so that changes can be made swiftly.
* Continuous testing from the start of development will help to avoid costly repairs later on or even after deployment.
* Validate against industry standard rules for security and risk.
* System should follow the modularity so that refactoring can be easy and fast.
* All code files should be accompanied by their respective unit test cases and code coverage report should be minimum 75%, though for some part/files it could be ignored based on valid justification from developer.
* Automation test framework will be implemented for System Integration Testing (SIT).
* All warnings from IDE will be considered as errors.
* All best practices should be followed during coding phase and proper code analysis check should be enforced by team leads and module owners.
* Code/Release branches merge should have proper code review mechanism.

# Non-Functional Requirement considerations

## Usability

* Speed of Use
  + Application user interface will enable end user an error and mistake free system operating.
  + Application will be available on desktop browsers (preferably google chrome) as well as on mobile device.
* Required User Ability
  + Application will be used by customer owning the policy, internal users, and partners.
* Learnability
  + Application will provide tour and video materials for users.
* Training Material
  + Application tour within the application will be available as training material.
* Language
  + Application will be implemented only in English.

## Reliability

* Maximum Failure Rate
  + Maximum failure rate should not be more than 1%. It must be in range of 0-1 %.
* Maximum Down Time
  + Acceptable system downtime is 0 per 24-hour period.
* Ease of Recovery
  + System will be auto recoverable. Assuming maximum acceptable time for restarting the system is 5 min after failure.
* Maximum Known Bugs
  + System or any part of the system is not acceptable with known high severity bugs.
  + System is acceptable with up to 5 known medium severity bugs with defined work arounds.
  + Medium and low severity bugs should be reported in release document with future action items so as to avoid them.

## Performance

* Throughput
  + Assuming 100 thousand concurrent users may use the system at any given peak-use time.
  + Assuming that 5 servers is acceptable under load balancer under peak time. Normally application will have two servers under load balancer.
  + The offered solution must complete 99% of provided services in less than 5000 milliseconds, over both the peak and non-peak hours.
* Response Time
  + Acceptable response time range 0-5 (in seconds) to a user action on the system.

## Security

* All the web API call is secure with JWT token and only authenticated and authorize use will access the system.
* Solution provided guard against following security threats.

1. [Injection](https://blog.sucuri.net/2018/10/owasp-top-10-security-risks-part-i.html)
2. [Broken Authentication](https://blog.sucuri.net/2018/10/owasp-top-10-security-risks-part-i.html)
3. [Sensitive data exposure](https://blog.sucuri.net/2018/10/owasp-top-10-security-risks-part-ii.html)
4. [XML External Entities (XXE)](https://blog.sucuri.net/2018/10/owasp-top-10-security-risks-part-ii.html)
5. [Broken Access control](https://blog.sucuri.net/2018/12/owasp-top-10-security-risks-part-iii.html)
6. [Security misconfigurations](https://blog.sucuri.net/2018/12/owasp-top-10-security-risks-part-iii.html)
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with known vulnerabilities
10. Insufficient logging and monitoring

## Supportability

* Ease of Installation
  + Solution is develop using docker. So, it is easily portable and deployable on any environment. Docker images published on all targeted registry using fully integrated CI\CD pipeline.
  + We are assuming that client is already on azure cloud and we want to deploy our multi container application on Azure Kubernetes Services.
  + YCompany DevOps engineer will do the deployment job of e-claim and we will be providing online support during this process.
* Planned Maintenance
  + Our targeted deployment environment is self-managed AKS. So, maintenance is very minimum, and everything will handle via browser.
  + YCompany DevOps engineer are responsible.
  + Assuming monthly maintenance and release cycle.
* Backup
  + Back up will be handled by azure cloud service.
  + No user interaction is required for backup.
* Ease of Configuration
  + AKS cluster configuration is required for setting up the no of cluster and no of microservice instances.
  + Need to configure policies for increasing the number of instances of handling the high load.
  + Will be handled by IT admin of YCompany, having access of azure account.
* Ease of Testing
  + Application should be tested with manual as well as automated cases.
  + User acceptance testing is required at the end of development cycle.
  + Load testing will be required at the end of development cycle.

## Infrastructure Requirements

* Clients
  + Client system should have at least 1 GB RAM.
  + All latest and previous two versions are supported fully on desktop and mobile.
* Servers
  + We recommend deploying the application on azure AKS. If we want to deploy on premises Kubernetes cluster. We recommend having at the three clusters to deploy all microservices spread on all the clusters.
  + Each cluster machine should have 4 GM RAM and 8 core processor and 100GB hard disk.
* Networks
  + 100 MBPS bandwidth network connection is required by Web Server and Application Server.
* Web Services
  + SMS Provider service
  + Mail Exchange service
  + Azure file storage service
* Environment
  + Application is multi container application and will run in on-premises or AKS.

# Solution Architecture

Solution architecture is based on the Microservice architecture pattern which requires implementation of an API gateway as the single-entry point for all clients. The API gateway handles requests in one of two ways. Some requests are simply proxied/routed to the appropriate service. It handles other requests by fanning out to multiple services.

We have defined a separate API gateway for each kind of client and this type of architecture pattern is known as **Backends for frontends**. In this architecture style, there are three kinds of clients: web application, mobile application, and internal application like bulk upload or back office. There are three different API gateways. Each one is providing an API for its client.

Using an API gateway has the following benefits:

* Insulates the clients from how the application is partitioned into microservices.
* Insulates the clients from the problem of determining the locations of service instances.
* Provides the optimal API for each client.
* Reduces the number of requests/roundtrips. For example, the API gateway enables clients to retrieve data from multiple services with a single round-trip. Fewer requests also mean less overhead and improves the user experience. An API gateway is essential for mobile applications.
* Simplifies the client by moving logic for calling multiple services from the client to API gateway.
* Translates from a “standard” public web-friendly API protocol to whatever protocols are used internally.

The API gateway pattern has some drawbacks:

* Increased complexity - the API gateway is yet another moving part that must be developed, deployed, and managed.
* Increased response time due to the additional network hop through the API gateway - however, for most applications the cost of an extra roundtrip is insignificant.

## Solution diagrams

### Solution Design

**Docker Host or AKS**

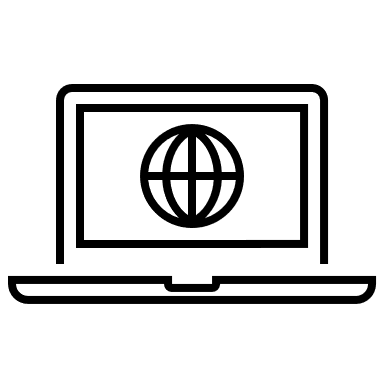
**Event bus**

**Identity Microservice**

Web API

**SQL Server**

API Gateways



**SQL Server**

Web API

**Claim Microservice**

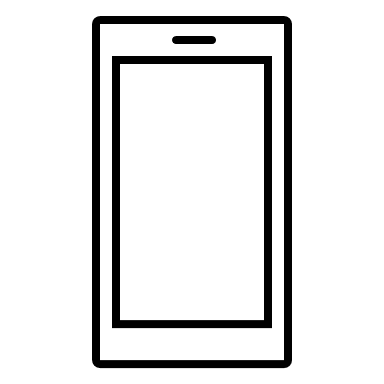
Aggregator

Web

API

Gateway

INGRESS (NGINX)

`

SMS

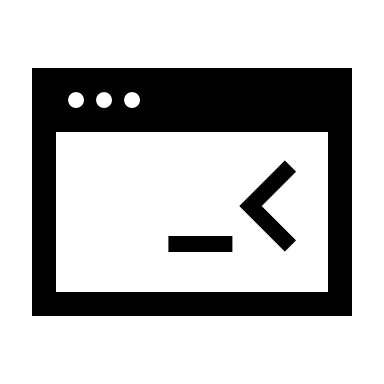
Web API

**Notification Microservice**

Aggregator

Mobile API Gateway

Exchange



**Insurance Microservice**

Aggregator

Internal app API Gateway

Web API

**SQL Server**

Web API

**Vehicle Rental**

**Microservice**

**SQL Server**

Web API

**Partner Workshop**

**Microservice**

Web API

**Document Store**

**Storage Microservice**

**SQL Server**

Web API

**Payment Microservice**

**SQL Server**

#### Components description

##### Client

Client component is implemented using the angular framework. This is a web application which provide the interactive screen to user to communication with the application. This component is hosted on NGINX web server. It is the entry point of Electronic Claim Processing Application. It provides following feature.

* Interactive interface to user.
* Perform validation on users input.
* Perform web API calls to perform operations.

##### Ingress/NGINX

Ingress/NGINX is a middle tier between the Internet and API Gateways, acting as a reverse proxy. API Gateways are front-ends or façades surfacing only the services but not the web applications (SPA). Ingress is also responsible for serving static content directly and redirecting HTTP API requests to API Gateway.

##### API Gateway

When we design, build large or complex microservice-based applications with multiple client apps, a good approach to consider can be an API Gateway. API Gateway pattern is the Backends for frontends pattern. It defines a separate API gateway for each kind of client.

Using an API gateway has the following benefits:

* Insulates the clients from how the application is partitioned into microservices
* Insulates the clients from the problem of determining the locations of service instances
* Provides the optimal API for each client
* Reduces the number of requests/roundtrips. For example, the API gateway enables clients to retrieve data from multiple services with a single round-trip. Fewer requests also mean less overhead and improves the user experience. An API gateway is essential for mobile applications.
* Simplifies the client by moving logic for calling multiple services from the client to API gateway
* Translates from a “standard” public web-friendly API protocol to whatever protocols are used internally

**Backends for frontends:**

Backend for frontends:

Web Applicaion

Internal App Gateway

Web App Gateway

Mobile App Gateway

Mobile App

Internal Application

Web Application

**API Gateway under the hood:**

API

Gateway

Service

**1**

**2**

**3**

**4**

**5**

**6**

**7 Other Middleware**

**8**

**Authentication & Authorization**

**Middleware**

##### Identity Microservice

Identity microservice or authentication microservice is only responsible for authenticating and authorizing the user. This service is implemented using the .net authentication provider. Its supports customer, incident manager, regional incident manager, adjuster, partner, report manager roles. It is based on JWT authentication and it has separate user database. Authentication and authorization are done by integrating identity server with Ocelot API gateway.

When accessing the Ocelot gateway, we need to achieve the characteristics of internal and external isolation, so we hosted Identity Server services to the Ocelot gateway, so that we can unify the access of authentication and service requests.

Following diagram describe the flow of authentication and authorization:

Token

Authorised?

Service info & Token

Response

Request + Token

Credentials

Token

Credentials

Response

Request

Ocelot

Microservice

Identity Server

Client

##### Claim Microservice

Claim microservices is implemented using the asp.net core web API. It exposes rest end point related to claim and communicate with other microservice using asynchronous event bus. When user report the incident, we generate the claim id. This claim id is used for further communication.

##### Notification Microservice

Notification microservice provide the functionality for sending the email and SMS on mobile. It communicates with exchange server for sending email and SMS provider for sending the SMS on mobile. It can get the request for sending the mail and SMS through web API end point or through event bus.

##### Storage Microservice

Storage Microservice provide the functionality to store the claim document in centralize document store.

##### Partner Workshop Microservice

Partner workshop microservice provide the list of partner workshop and it also provide the functionality of booking the appoint with workshop.

##### Payment Integration Microservice

Payment integration microservice provide the functionality to pay the bill online. There are two features of this service.

* Payment made by policy customer to pay workshop related fee and adjustments as per insurance policy.
* Payment made by YCompany to workshops for the settlements.

##### Vehicle Rental Microservice

Vehicle Rental microservice provide list of car rental providers and available cars, details regarding the same.

# Framework/Infrastructural Components

## Logging

Logging is an important aspect of application. .NET Core supports a logging API that works with a variety of built-in and third-party logging providers. We are considering azure cloud for hosting and deployment the e-claim application. So, we will use Azure Application Insights provider and it stores logs in Azure Application Insights. For on premises deployment we will use the event source provider.

## Exception Handling

Electronic Claim processing application will have two type of exception handling

1.Global Exception Handling: Any unhandled exception will be captured by this module and exception will be handled gracefully and response will be sent according to exception. In most of the cases response to client application will be Internal Server Error.

2.Exception Handling: Each class in each layer will handle the exception and log the exception for future use. After handling the exception gracefully, function/module will return value in way so that client will take right action.

## Security

Electronic Claim Processing system is secured with JWT authentication. Each Web API service are configured with authentication and authorization filters so only valid user with sufficient permission can access the services. Apart from this REST API only available on secure network. Security is implemented in API Gateway.

## Data Access

Electronic Claim Processing application will use Sql Server as the database and Entity framework Core as ORM between application and database. Application will leverage the code first approach and each entity in the application will have corresponding table in the database. Application will have module to handle the migration of database.

## Input Validation

Input validation will be implemented at two places.

1. Angular client – Angular input validation will be used to give quick response to user about the input data.

2. API Model Validation- Dot Net provide the attribute-based validation so our application will use the same and if any specific validator does not exist in asp.net then we will create our custom validators. If input validation will fail, then API will return the bad request response with information about missing/wrong data.

## Object Creation

We want to keep our application loosely coupled so we are going to implement the dependency injection. Application will leverage Dot Net Core in-built dependency injection framework.

## Microservices communication

As mentioned in solution diagram we will be implementing asynchronous event-driven communication, one microservice publishes events to an event bus and many microservices can subscribe to it, to get notified and act on it. Our implementation will be based on AMQP protocol to use for event-driven, message-based communications.

When we use an event bus, we might want to use an abstraction level (like an event bus interface) based on a related implementation in classes with code using the API from a message broker like RabbitMQ or a service bus like Azure Service Bus.

# Technology Stack

|  |  |  |
| --- | --- | --- |
| SN | **Component** | **Recommendation** |
| 1 | Web Client | Angular 10 with Typescript |
| 2 | Mobile Client | NativeScript (for Native Application) |
| 3 | Services | Microsoft Visual Studio 2019 IDE  Dotnet Core 3.1  ASP.Net core  SignalR  JWT Authentication |
| 4 | Storage Management | Azure File Service |
| 5 | ORM | Entity Framework Core |
| 6 | Database | SQL Server 2008 or later |
| 7 | Application Hosting | On-premises docker container, Azure Kubernetes Service |
| 8 | Reporting | SSRS |
| 9 | Continuous Integration | Jenkins Pipeline |
| 10 | Repository | Bitbucket |
| 11 | Code quality | Code Analysis |

# Glossary

| Glossary Term | Definition |
| --- | --- |
| API  JWT  IDE    AMQP | API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.  JSON Web Token is an Internet standard for creating data with optional signature and/or optional encryption whose payload holds JSON that asserts some number of claims. The tokens are signed either using a private secret or a public/private key.  An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.  The Advanced Message Queuing Protocol (AMQP) is an open standard for passing business messages between applications or organizations. It connects systems, feeds business processes with the information they need and reliably transmits onward the instructions that achieve their goals. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Revision History**  **COPYRIGHT NOTICE**  Copyright © 2013-17 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. | | |  |
| Version | Date | Author/Contributor | Comments |
| 1 | 26-01-2021 | Pankaj Mahur | Initial draft for solution design document |
|  |  |  |  |