Hatteland

Welcome!
This is the home page for your team space within Confluence. Team spaces are great for sharing knowledge and collaborating on projects, processes and procedures within your team.

Next, you might want to:

- **Customise the home page** - Click "Edit" to start editing your home page
- **Create additional pages** - Click "Create" to choose a blank page or template
- **Write a blog post** - Click "Create" and select "Blog Post" to share news
- **Manage permissions** - Click "Space Tools" and select "Permissions" in the sidebar to manage what team members see

The team

@ Samudra Kanankearachchi

@ Raashid Ahamed

@ Hashan Wanniarachchi

@ Sachith Jayasinghe

@ Warsha Kiringoda

@ Sahan Jayasinghe

@ Obhasha Priyankara

@ Fazna Fouseen

@ Yasiru Wickramasinghe

@ Raveen Dassanayake

@ ThavinduN

@ Dharana Rodrigo

@ YasiruS

@ SrilalS

About us

Team space to share documentation of team activities

Blog stream

Blog stream

Create a blog post to share news and announcements with your team and company.

Create blog post

Recently updated

PHR Policy Suggestions with GPT3 Models
Feb 28, 2023 • contributed by SrilalS

Srilal Siriwardhana
Feb 03, 2023 • contributed by Hashan Wanniarachchi

Hatteland
Feb 03, 2023 • contributed by Hashan Wanniarachchi

Career Development Plan
Jan 11, 2023 • contributed by Hashan Wanniarachchi

Onboarding security checklist
Dec 09, 2022 • contributed by Warsha Kiringoda

Personal data journey map
Nov 04, 2022 • contributed by Warsha Kiringoda

General Induction
Oct 27, 2022 • contributed by Warsha Kiringoda

Information security on transit
Oct 09, 2022 • contributed by Sahan Jayasinghe

Information Security Principals
Oct 07, 2022 • contributed by Fazna Fouseen

2022 - H2
Oct 06, 2022 • contributed by Sachith Jayasinghe

Hatteland Domain Knowledge Sharing

Getting start rambase app development

Video on how to do rambase app development

https://onedrive.live.com/?authkey=%21AJBlPXCdsgYupVI&cid=D69F53D31FAC388F&id=D69F53D31FAC388F%21400365&parId=D69F53D31FAC388F%213156&o=OneUp

Common Root Causes for bugs

If you have found any recurring or common root causes, you can add them here. so we can take it up on team discussions to avoid further mistakes of same nature

Issue No	Root Cause	Suggestions to prevent	Discussed on	Status
RCF-5447	Adding rb-color-picker directive has changed the widget header structure	Check for click (and other) events which relates to the corresponding insertion point and it's immediate parents when testing for functionality	19/10/2020	
RCF-5555	Expression for Ng value was a static string	For Ng-Value, always use a scope variable	27/10/2020	
RCF-5582	Field size and Label size are used wrong	Define field/label size as "fieldsize", and not like field-size"	28/10/2020	
RHR-557	Ng-repeater list componnents dont have bookmark url set from client framework automatically.	Add a custom handling of bookmark URL (Refer LIB /4992).	23/10/2020	
RHR-692	Client framework doesn't support multiple instances of the same LIB in the same APP	If the second usage is coming from CMI create two LIBs. Otherwise, if multiple instances are used in the APP itself drop parameters to the app scope as committed in here.	10/10/2020	

CRM-2498 CRM-2131	In task application in board view, after updating the title of a task in details component, the title in the board was not updated. (autobind not working correctly).	execute a GET on the selected item Id. Earlier this was fixed by reading the entire datasource again, which caused the CRM-2498 bug in the board view.	2/17/2022	

Hatteland Spec Team Building

Hatteland Team Continuous Learning

1. Team should have a knowledge base preferably in the form of a wiki with live documentation maintained in it.
2. Team should establish mechanisms to continuously acquire domain knowledge from the onsite counterparts.
3. Information regarding day-today activities of the team is continuously shared across everyone in the distributed team in real-time.

Hatteland Team Culture

1. Internal team satisfaction survey is conducted quarterly and results are discussed within the team to take improvement actions
2. All team members are aware of cross cultural aspects. Members are able the explain the means of neutralizing such differences in a distributed work environment.

Peer Feedback

Team conducts offline surveys/assessments (possibly anonymous) where every member would provide feedback to others in the team at least quarterly.

Hatteland Tech Talks

Connecting Bitbucket and Jenkins (through SSH)

In a previous post we mentioned about what is Continuous Integration and how to configure one such tool, Jenkins. There when configuring a new job we had a setting where we have to specify the git account in order to get the source code from. Today lets talk about how to specify credentials to link this git account(bitbucket) with Jenkins.

Pre-requisites

- Jenkins server installed.
- Git installed.
- Bitbucket account.
- Bitbucket repository.

How to Link Bitbucket with Jenkins

Generate SSH key

1. Open Git Bash terminal.
2. Enter the following command to verify the SSH client is available:
   ```
   $ ssh -v
   ```
3. If you have installed ssh the following will be displayed. If not you have to install ssh via your package manager.
   ```
   OpenSSH_4.6p1, OpenSSL 0.9.8e 23 Feb 2007
    usage: ssh -1246AaCfgkMNnqsTtVvXxY-b bind_address-c cipher_spec
    [-D bind_addressport] -e escape_char-F configfile
   -i identity_file [-L bind_addressport:host:hostport]
   -l login_name-m mac_spec-O ctl_cmd-o option-p port
    [-R bind_addressport:host:hostport] -S ctl_path
    [-w local_tunremote_tun] user@hostname commandEnter ssh-keygen at the command line.  The command
   prompts you for a file to save the key in:
   ```
4. List the contents of your `~/.ssh` directory.
   If you have not used SSH on Bash you might see something like this:

   ```
   $ ls -a ~/.ssh
    ls: /c/Users/janithat/.ssh: No such file or directory
   ```

If you have a default identity already, you'll see two `id_*` files:

```
$ ls -a ~/.ssh

. .. id_rsa id_rsa.pub known_hosts
```

Now We can create the ssh keys.

1. Enter `ssh-keygen` at the command line.
   The command prompts you for a file to save the key in: (Press enter to save in default location – `c/Documents and Settings/(user)/.ssh/id_rsa`)
2. Enter and renter a pass phrase when prompted. (Do not enter a password)
3. The command creates your default identity with its public and private keys. The whole interaction looks similar to the following:
   ```
   $ ssh-keygen
   Generating public/private rsa key pair.
   Enter file in which to save the key (/c/Users/(user)/.ssh/id_rsa):
   Created directory '/c/Users/(user)/.ssh'.
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /c/Users/(user)/.ssh/id_rsa.
   Your public key has been saved in /c/Users/(user)/.ssh/id_rsa.pub.
   The key fingerprint is:
   e7:94:d1:a3:02:ee:38:6e:a4:5e:26:a3:a9:f4:95:d4 manthony@MANTHONY-PC
   ```
4. `List the contents of ~/.ssh to view the key files. The following should be displayed.`
   ```
   $ ls ~/.ssh
   id_rsa  id_rsa.pub
   ```

Next step is to create SSH config file and update .bashrc profile.

1. Copy this config file to ~/.ssh folder (C:`/Documents and Settings/(user)/.ssh`)
2. `Then copy this .bashrc file to user folder (C:/Documents and Settings/(user)/)`
3. `Close gitBash.`
4. `Reopen gitbash.`

Now verify the that the script identity added your identity successfully by querying the SSH agent issuing ssh-add -l command. Something similar to following should display.

```
2048 0f:37:21:af:1b:31:d5:cd:65:58:b2:68:4a:ba:a2:46 /Users/(user)/.ssh/id_rsa (RSA)
```

Now we have created the ssh keys. The next step is to add the public key to Bitbucket account.

1. Log on to Bitbucket account.
2. Choose avatar –> Mange Account
3. Click **SSH Keys**.
4. In your terminal window, `cat` the contents of the public key (cat ~/.ssh/id_rsa.pub) file or open id_rsa.pub with notepad.
5. Select and copy the key output in the clipboard.
6. Enter a **Label** for your new key.
7. Paste the copied public key into the **SSH Key** field.
8. Click the **Add key** button.
9. Return to the terminal window and verify your configuration by entering the following command.
   `ssh -T git@bitbucket.org`
10. The command message tells you which Bitbucket account can log in with that key.
    ```
    conq: logged in as janithat.
    You can use git or hg to connect to Bitbucket. Shell access is disabled.
    ```

Next we have to add the private key to Jenkins in order give authentication for Bitbucket for Jenkins.

1. In Jenkins Dashboard click **Credentials.**
2. Click **Global credentials.**
3. Next click **Add credentials** which will open a form where you can add credentials.
4. From kind drop down list select **SSH username with private key**.
5. Type a user name and then copy the id_rsa key (C:`/Documents and Settings/(user)/.ssh/id_rsa`) or check From a file on Jenkins master and give the path. (C:`/Documents and Settings/(user)/.ssh/id_rsa`)

6. Click OK.
7. Now when you give project repo on Jenkins jobs you can select this private key to give authentication for Bitbucket.

Continuous Integration (CI) with Jenkins

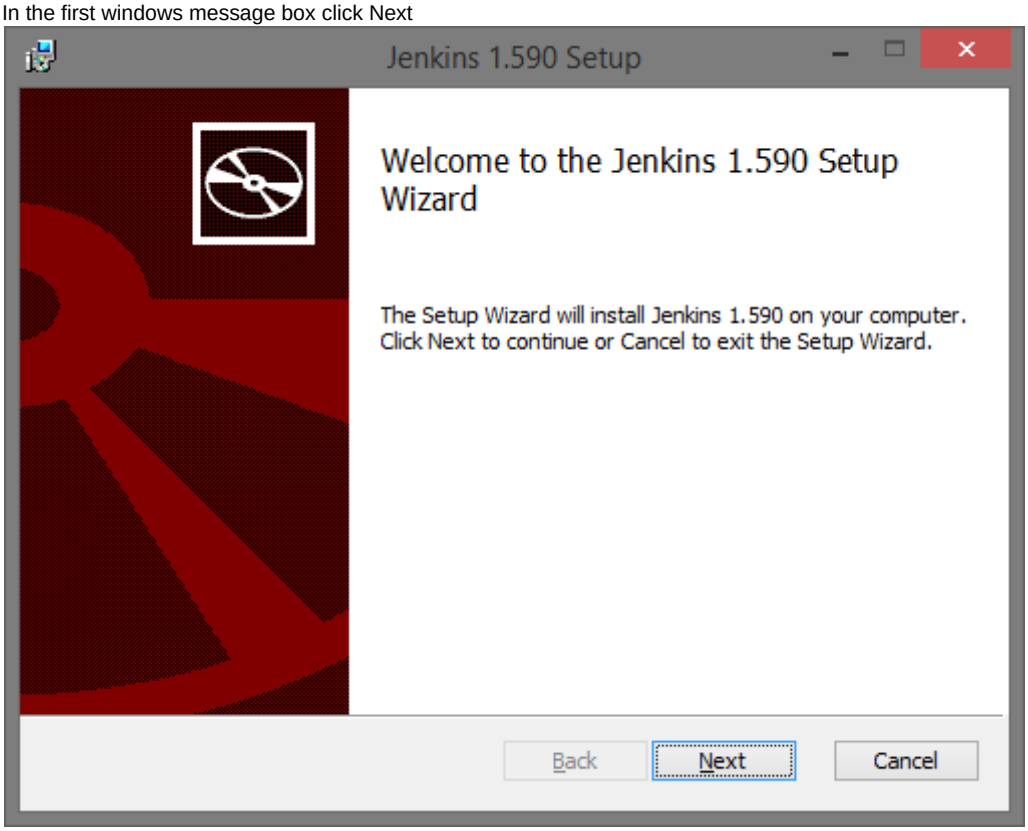
Introduction

The first question that we need to answer is what is ‘Continuous Integration’?. According to TechTarget “Continuous integration (CI) is a software engineering practice in which isolated changes are immediately tested and reported on when they are added to a larger code base. The goal of CI is to provide rapid feedback so that if a defect is introduced into the code base, it can be identified and corrected as soon as possible. Continuous integration software tools can be used to automate the testing and build a document trail. **Jenkins** is an open source continuous integration tool”. Since Jenkins has MIT licence it can be used for free in commercial environments as well, which is the most advantageous with regard to other CI tools available in the market.

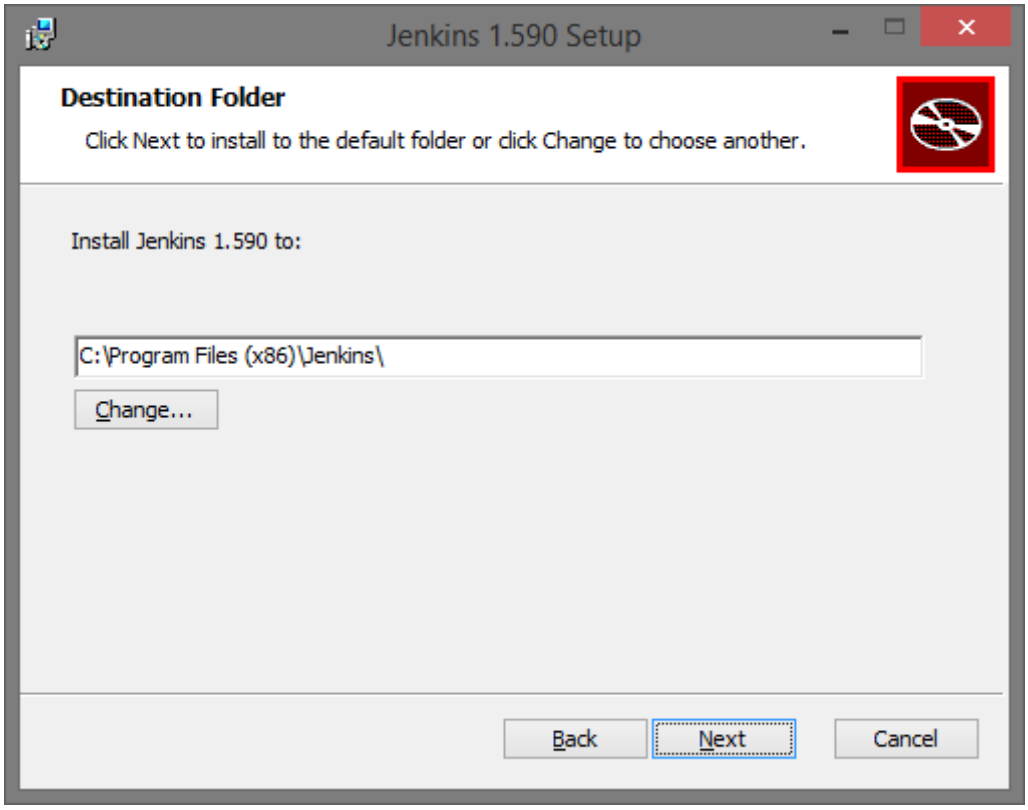
To know more about Jenkins you can visit their website from here. All the things that are need to know about this CI tool is specified there. So lets continue with our first topic. How to install this tool in to your production server. First we need to download Jenkins. Visit http://jenkins-ci.org/ t o download Jenkins. (Jenkins is available for all major OS platforms Windows, Mac, Linux)

How to install

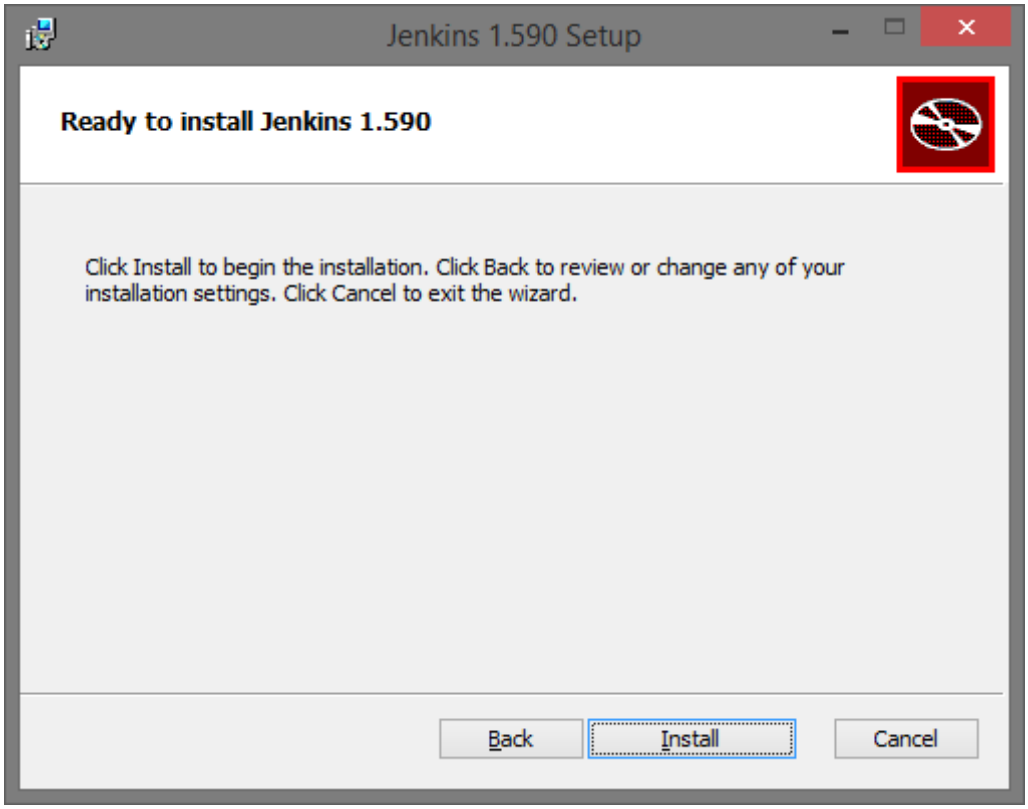
1. Extract the zip file and run jenkins.msi
2. In the first windows message box click Next



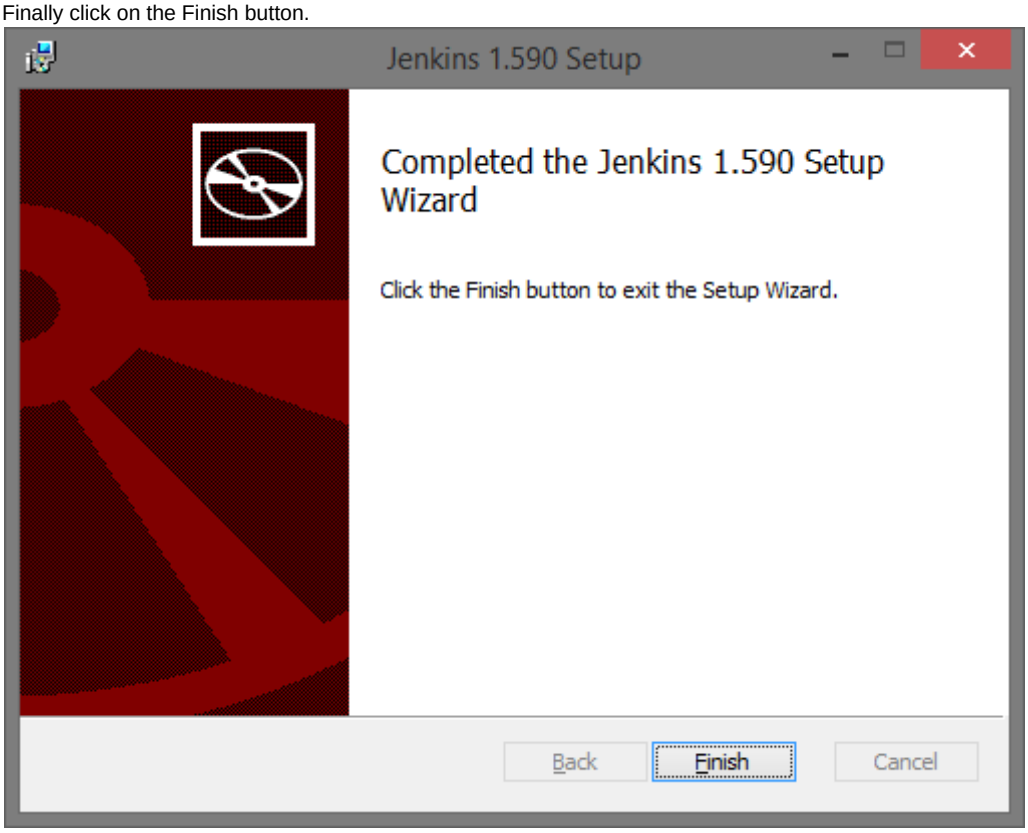
3. Choose the destination folder to install Jenkins.



4. Next click on Install button



5. Finally click on the Finish button.



Ok, now we have installed Jenkins in the machine. But before we start using continuous integration there are some configurations that are need to be done.

Setting up Jenkins

After installation Jenkins would run on your default browser path http://localhost:8080/. Here the port can be changed if you have another application already taken 8080 port or you are thinking of using 8080 port in the future.

Change Jenkins Port

1. Go to the directory where you installed the Jenkins (In default, it is under Program Files/Jenkins)
2. Open the Jenkins.xml
3. Search "–httpPort=8080" and replace the "8080" with the new port number that you wish.

Before we add a new project to Jenkins there are some configurations needed to be done. First we need to add git support to the Jenkins. Before adding it to the Jenkins we have to install git in our machine and then configure it on jenkins.

1. First go to http://git-scm.com/downloads
2. Install Git in your machine.
3. In Jenkins click on Manage Jenkins > Global Tool Configuration
4. And then under Git section give the git installation path. E.g.: "C:\Program Files (x86)\Git\cmd\git.exe"

After adding git support next we need to add MSBuild plugin which will help Jenkins to build the project.

1. Click on Manage Jenkins > Manage Plugins
2. Then switch to Available tab
3. Search for MSBuild Plugin
4. Select the plugin and click on Install without restart

After installing all the prerequisites now we can configure our project with Jenkins. Ok so now we are going to create our first job.

To create a new project

1. Click on Jenkins > New Item
2. Type the item name
3. Tick on Freestyle project
4. Click on Next

And now in Configuration page go to Source Code Management section.

1. Select Git as the type of source control.
2. Provide the repository url “git clone git@bitbucket.org:janitha000/citesting.git”
3. And select a credential type(Note:- To complete this step you should have added Credential. (here we need to configure our git account with Jenkins in order to give authentication)
4. Next select a Build Trigger (Select Poll SCM)
5. Then add a build step Build a Visual Studio project or solution using MSBuild
6. Next give the MSBuild file path “..\Documents\Visual Studio 2013\Projects\SampleProject\SampleProject.sln”
7. Finally click on Save

And now you have configured your project with Jenkins..

Getting started with TypeScript

Introduction

TypeScript is an open-source language developed by Microsoft. TypeScript (TS hereafter) is basically JavaScript(JS hereafter) with optional typing and many other useful additions. TS is a superset of Ecmascript 5 (ES5) and incorporates features proposed for ES6. TS is a compiled time language, which gets compiled to JS. TS compiler takes ***.ts*** files and compiles them in to ***.js*** files. There’s been a lot of hype around the subject as of late. Let’s explore why that is….

Installation

Visual Studio

In VS 2013, you have to install the TypeScript Plugin for TS to work, but in VS 2015 this comes pre-installed. WebEssentials will give a nice split view of the compiled JS file if you opt to install it for VS 2013 too. But sadly, webessential support for TS is discontinued in VS 2015, at least for now. If you can’t see the compiled code on VS 2013, go to “Tools -> Options -> Text Editor -> TypeScript -> Project” and select the option to “*Com pile-on-save*”.

Using Node.js

You need to have Node.js and NPM installed in your machine prior to this. If not, check here for installation details. After Node and NPM is installed, you can install and use TS using following commands.

Installation

```
npm install -g typescript
```

Command Line Usage

tsc is the TS compiler & the command for compiling, running and introducing watchers for auto-compiling.

Compile

```
tsc app.ts
```

Multiple file compile

```
tsc app.ts another.ts someMore.ts
```

Wildcards

```
tsc *.ts
```

Joining

```
tsc *.ts --out app.js
```

Watchers

Instead of having to run ***tsc*** each time, we can use ***- -watch*** to auto compile the specified TS file(s) to JS files whenever there's a change.

```
tsc *.ts --out app.js --watch
```

A Few Basic Features To Get You Started…

Optional Typing

In Contrast to JS, we can provide explicit types to the variables in TS. For an example;

```
var myName: string = "Andrew";
```

And this can be used with function parameters too!.

```
function printName(name: string) {
    console.log(name);
}
```

By using this typing option, we get warnings if we try to assign/pass incompatible data types to the respective variable(s). Also, it helps with ***auto-completion***.

Interfaces

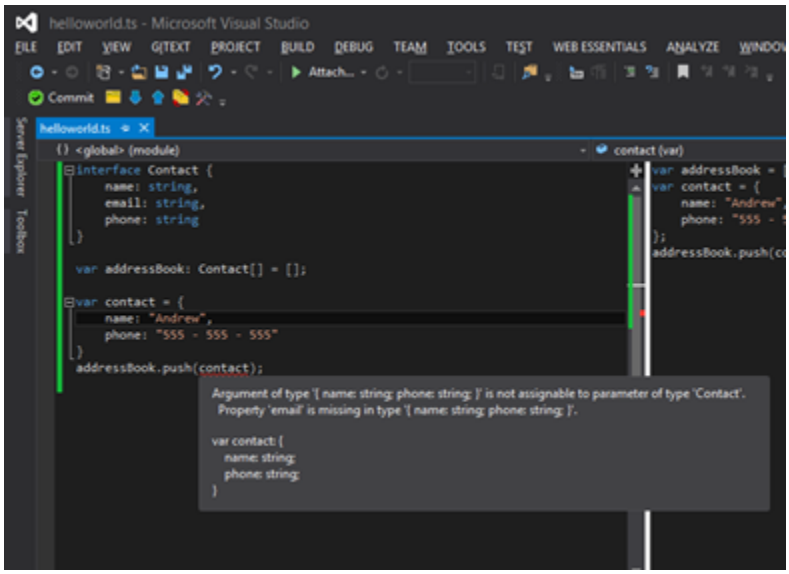
We can define the shape of object literals using interfaces. It's merely enough to know the structure of the object as a data store most of the time.

```
interface Contact {
    name: string,
    email: string,
    phone: string
}
var addressBook: Contact[] = [];
```

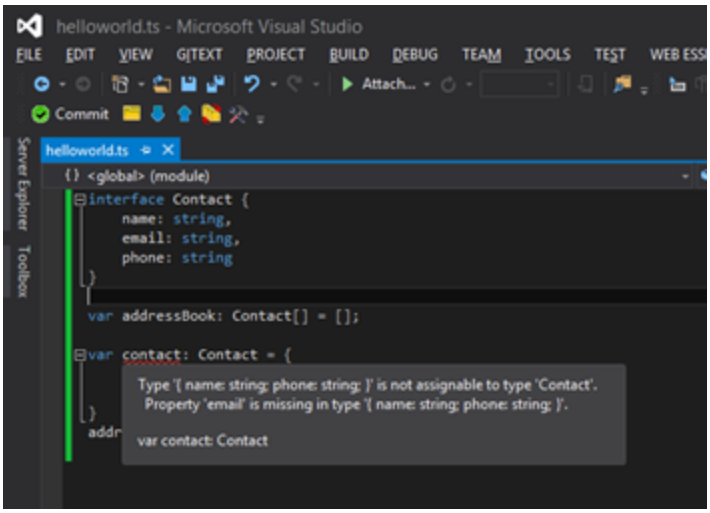
We can push a ***Contact*** object to the list as follows;

```
var contact = {
        name: “Andrew”,
        email: “andrew@yourmail.com”,
        phone: “555-555-555”
}
addressBook.push(contact);
```

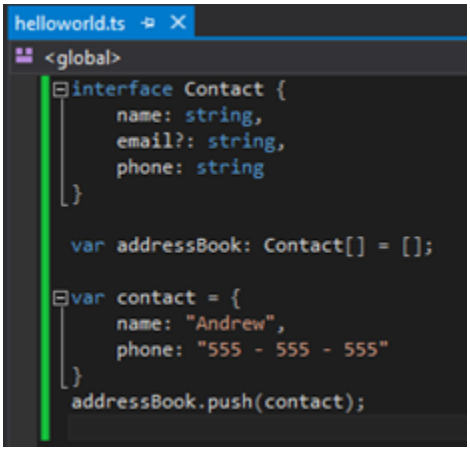
If we define ***“contact”*** object with properties missing(such as ***email***), the editor will give us an error saying *“contact”* cannot be assigned to the type ***Contact***.



We can also use the type when we are initializing the “*contact*” object. It would give us the same error if we try to omit properties defined in the interface.



However, we can instruct TS to ignore the missing ***email*** property of the ***Contact*** type by using the "**?**" literal to indicate that the property is **optio nal**.



Arrow Functions

One of the interesting new parts of ECMAScript 6 are ***arrow functions***. These functions use an "arrow" syntax (**=>**) when defining functions. These arrow functions behave slightly different from a traditional JS functions in several ways;

- **Lexical** *this* **binding** – value of *this* is determined by the point where the function is defined, and not where it's being used.
- **Cannot create instances** – Arrow functions cannot be used as constructors.
- **Single** *this* – Value of *this* inside the function cannot be changed.

Hence, arrow functions will make it easier for us to deal with the confusing "*this*" scoping. And since these functions can only cater a single "this" value, it's easier for the JS engine to optimize the operations.

Basic syntax of the function looks like;

```
var sum (num1, num2) => { return num1+ num2; } //or
var sum (num1, num2) => num1+ num2
```

Which will be equivalent to;

```
var sum = function(num1, num2){

        return num1 + num2;

}
```

Parameterless arrow functions will take the form of;

```
() => {statements;}
```

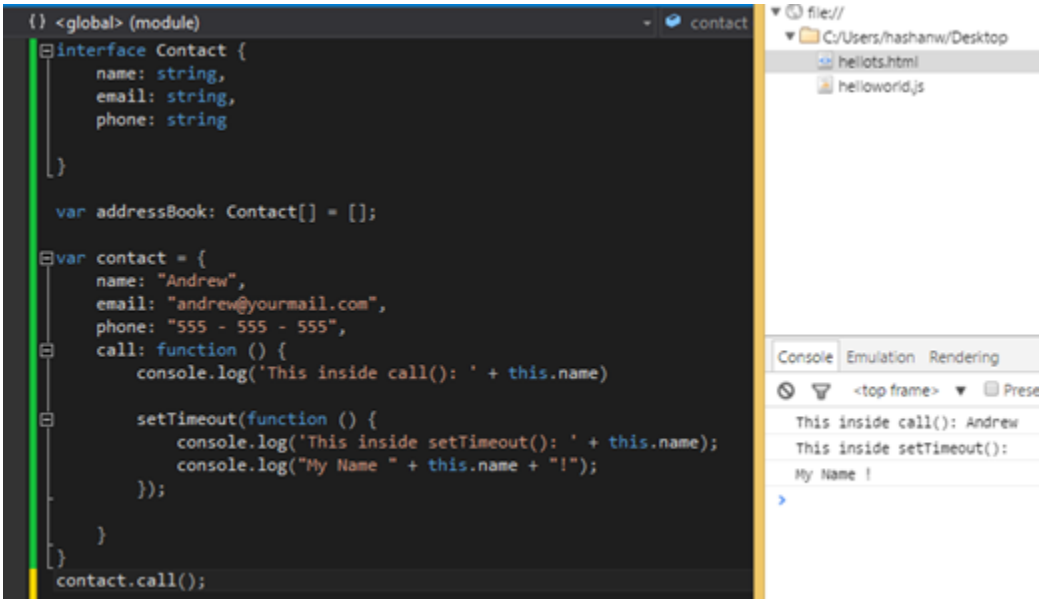
We can use these arrow functions through TS. For more information on how arrow functions work, refer here and here.

Let's consider the following code fragment;

```
var contact = {

    name: "Andrew",

    email: "andrew@yourmail.com",

    phone: "555 - 555 - 555",

    call: function () {

        console.log('This inside call(): ' + this.name);

        setTimeout(function () {

            console.log('This inside setTimeout(): ' + this.name);
```

```
            console.log("My Name " + this.name + "!");
        });


    }
}
contact.call();
```

Here, using ***this*** inside of the **setTimeout** function will cause in referring to the global **window** object instead of the **contact** object. So, ***this.name*** will evaluate to be empty.

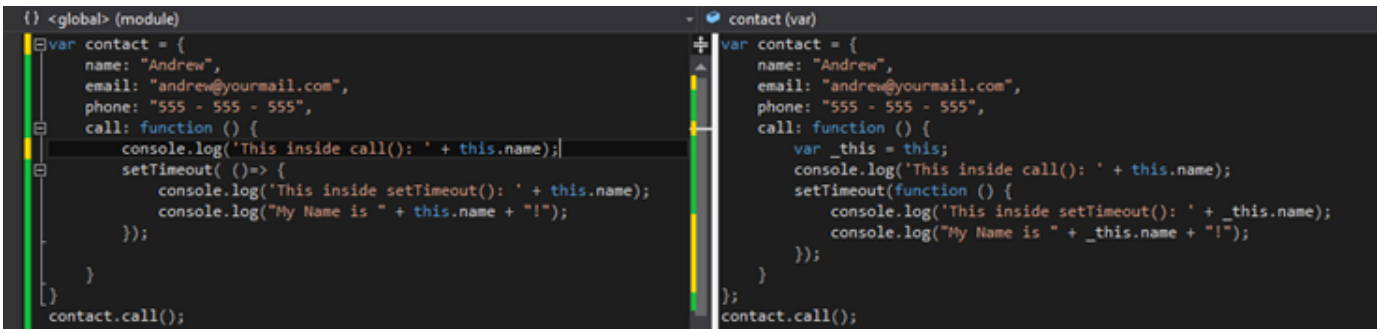


To Resolve this, we can use an arrow function inside the setTimeout.

If we take a look at the generated JS file, we'll see that the compiler injected a new variable,

```
var _this = this;
```

and used it in **setTimeout**'s callback function to reference the **name** property.



Insertion of this new "***_this***" variable ensures the correct reference to **this** runtime is used inside the ***setTimeout*** function.

Classes

ECMAScript6 introduced the "class" concept to front end scripting, which we are very familiar with when dealing with back-end technologies. Let's take a look at the class syntax support of TS.

Syntax

TS classes start with "***class***" prefix. However, the body of the class has some similarity to a function. It mainly contains a set of statements and functions. A special "***constructor***" function is used to initialize the class when it is instantiated. Let's consider the following code fragment.

```
class Contact {
    name: string;
    constructor(_name: string) {
        this.name = _name;
    }
    call() {
        console.log("Hi, My Name Is: " + this.name);
    }
}
var c1 = new Contact("Andrew");
c1.call();
```

This will print out the name in the console when the "call" method is invoked. With the ability to define classes, our next impression is to apply OOP concepts such as ***Encapsulation*** and ***Inheritance*** into these classes. Need not worry, TS helps us in using these OOP concepts too.

Encapsulation

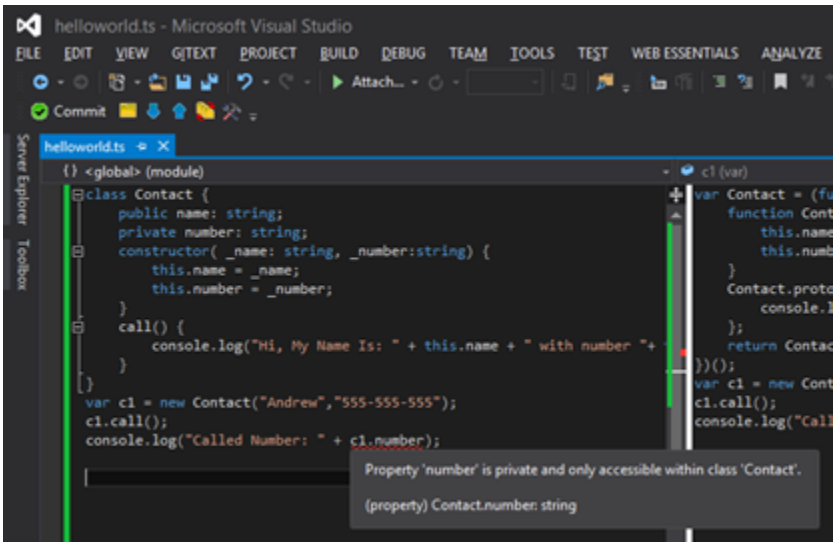
In JS, we are not much familiar with using "***public***" and "***private***" keywords to modify variable access levels. Instead, *var*, *this* and local variables were used. In TS, these keywords are supported and we can directly use them to restrict access to class properties and methods.

Let's revisit the Contact class. Assume you want to include a **number** property, which is a private variable. We will keep **name** as a public property. The changed class will look like;

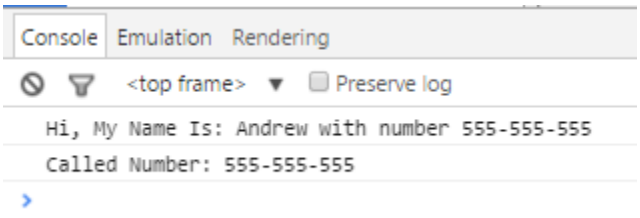
```
class Contact {
    public name: string;
```

```
    private number: string;
    constructor( _name: string, _number:string) {
        this.name = _name;
        this.number = _number;
    }
    call() {
        console.log("Hi, My Name Is: " + this.name + " with number "+ this.number);
    }
}
var c1 = new Contact("Andrew","555-555-555");
c1.call();
```

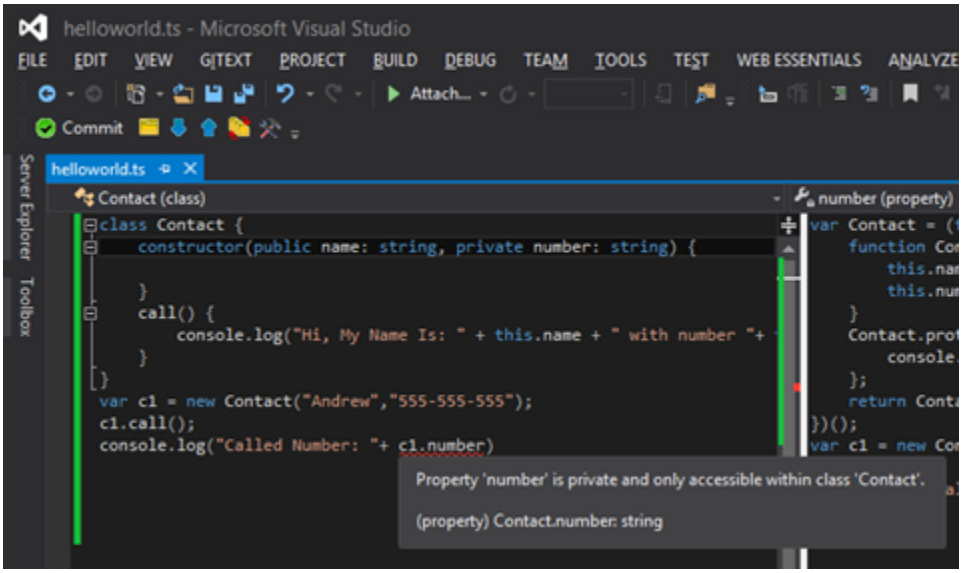
Notice the use of **public** and **private** keywords before the property declaration. Here, it will print the provided number in the console. However, if we try to access the number property of the ***c1*** object, TS will show us an error saying "***number***" property is private.



However, it will have no effect on running the application as JS doesn't have support to provide privacy. Hence, these encapsulation tactics will only be enforced during the *compile-time* and gives us errors and warning about possible privacy violations as shown above. When we run this code, it will actually print the number of the ***c1*** object we created, without giving any *run-time* errors.



Another way to declare these public and private properties is to provide these access modifiers directly to the constructor arguments. Any argument in the constructor with "***public***" or "***private***" keyword will be included as a property of the respective class with the corresponding access modifier.



This is a minimalist way in declaring class properties. However, EC6 prefer if we declare the properties beforehand, but it's not a strict rule.

Inheritance

As all our contacts are people, let's declare a base class named ***Human*** to inherit our ***Contact*** class. Human class has a public ***name*** property and a ***speak*** behavior related to it.

```
class Human {
    constructor(public name: string) {
    }
    speak() {
        console.log("Hi, My Name Is " + this.name + " and I'm a Human!!!");
    }
}
```

Now let's inherit our ***Contact*** class from the newly created ***Human*** class. Keyword ***extends*** is used to indicate the inheritance. The syntax is;

```
class childClass extends parentClass {
}
```

Inherited Contact class will look like;

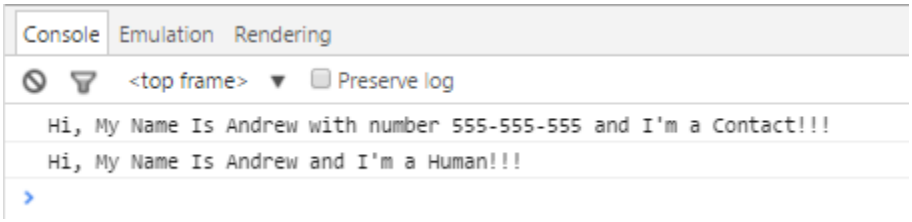
```
class Contact extends Human {
    constructor(public name: string, private number: string) {
        super(name);
    }
    speak() {
        console.log("Hi, My Name Is " + this.name + " with number "+ this.number +" and I'm a Contact!!!");
    }
```

```
    superSpeak() {
        super.speak();
    }
}
```

We have called the parent's constructor inside the constructor of the Contact class. ***speak*** method is *overridden* with our own method, but we can still access the parent's speak method and other public properties through the ***super*** keyword.

Now, let's instantiate the ***Contact*** class. When we invoke both of these functions, both implementations will be executed.

```
var c1 = new Contact("Andrew","555-555-555");

c1.speak();

c1.superSpeak();
```



End of the beginning

As you have seen, **TypeScript** is a new front end scripting language which offers much more functionality than JavaScript as of now. It's fun to learn and to try it ourselves. This small intro only covered a small part of its many, many features. But this will be a good point for you to start working with TS. Hope this helped, Happy Coding!

References

1 - Getting Started With TypeScript - treehouse

2 - Getting Started With TypeScript - tutsplus

Real Time Web with socket.io

Before going deep in to the technical implementation part let's first dive into the question of "What is Real Time Web?". In order to answer that we need to have an understanding of how clients and servers work. Even an amateur level software developer now knows what is client-server architecture model. If you don't know about client-server architecture model let me give a simple understanding about what it is because it is essential to know about it to continue in this article. Client-server architecture also knows as two-tier architecture is according to webopedia " *is a network architecture in which each computer or process on the network is either a client or a server". So* in the client-server architecture, the client sends the requests over the Internet using network communication and then the server will process that request and send the response back to the client. Here you can see that it is the client that start off the communication with the server. So client first initiates the communication and server responds for the request sent by the server. So what if the server wants to start off the communication and push responses without client requesting them first? That is where the term real time web comes into the play.

Real-time web is where the server gets the ability to push to the clients without client requesting for data first. Assume a chat application where two chat clients can communicate via a server. In this situation, it is a waste if all the chat clients request data from the server like every 1 second. Because some times there can be a chat during this 1 second or there will be another chat in like 10 minutes later. What is more efficient is the server to send data to client chat applications when a chat is received. This functionality can be done through real-time web application development. This is only a simple example on usage of real-time web functionality.

Ok now since we have the understanding of what is real time web lets talk about how we can implement this. There are many frameworks at present that support real-time communication. A described set of technologies in this context can be found in this site. Out of these technologies, the most popular technologies at current is SignalR and socket.io . SignalR is developed by ASP.NET and later became open source can be used in .NET also with javascript. The core of SignalR framework is written in c#. Since it is now an open source framework the source code can be seen in this link if you are interested. Socket.io in the other hand is written in JavaScript and currently can be used with JavaScript/Java and c++ applications. Since it is also an open source framework the source code can be seen here. Both are really good frameworks and I have worked with SignalR for more than 2 years. The disadvantage in socket.io is that the SignalR has very good getting started documentation where socket.io does not have. That is why I am writing this post using socket.io rather than using SignalR. If you are interested on SignalR also here is the link for their documentation.

So how does these frameworks achieve real time web functionality? To answer that question we have to know little bit on network transport methods that are used. There are four methods that both SignalR and Socket.IO uses.

HTML 5 transports

These are transports that are supported by HTML 5. These transport methods can further divided in to two groups.

- **WebSocket** - WebSocket is the only transport that establishes a true persistent, two-way connection between client and server. However, WebSocket also has the most stringent requirements; it is fully supported only in the latest versions of Microsoft Internet Explorer, Google Chrome, and Mozilla Firefox, and only has a partial implementation in other browsers such as Opera and Safari.
- **Server Sent Events -** also known as EventSource (if the browser supports Server Sent Events, which is basically all browsers except Internet Explorer.)

Comet transports

These are transports which are based on the Comet web application model, in which a browser or other client maintains a long-held HTTP request, which the server can use to push data to the client without the client specifically requesting it. There are two methods in this transports.

- **Forever Frame** - Forever Frame creates a hidden IFrame which makes a request to an endpoint on the server that does not complete. The server then continually sends script to the client which is immediately executed, providing a one-way realtime connection from server to client. The connection from client to server uses a separate connection from the server to client connection, and like a standard HTML request, a new connection is created for each piece of data that needs to be sent. Only Internet Explorer supports this transport method.
- **Ajax long polling**. Long polling does not create a persistent connection, but instead polls the server with a request that stays open until the server responds, at which point the connection closes, and a new connection is requested immediately. This may introduce some latency while the connection resets.

As you can see Web Sockets are the only method that can support establishing a true persistent, two-way connection between client and server. So why can't we just implement real time web applications using Web Sockets rather than using different frameworks for it. The problem is that not all browsers or machines supports Web Socket transport. This problem is what these frameworks addresses. When starting a connection it automatically identifies the most suitable transport and uses it. So in SignalR it starts the communication with Web Sockets and if it is not supported it will fall back to SSE, FF and long polling respectively. For the Socket.IO it is the other way around where it first starts the communication with long polling and go higher using FF, SSE and Web Sockets. By using these frameworks the developer does not need to concern about what are the transport methods are supported by the clients and servers because they are automatically handled.

Ok since now we have a pretty good understating on what is real time web and how real time web works lets go in to developing a simple real time web application. In this post let's try to build a chat application using Socket.IO. First we need a server, so let's create a server using Node.js and express framework. If you want to know detailed information on how to create these servers you can look in to my previous posts on Creating Web Servers using Node.js and Creating RESTful Web Services using Node.js.

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
res.send('Hello World');
})

var server = app.listen(8081, function () {
console.log("Server listening at port 8081");
})
```

Above code represents a simple server created using node.js and express framework. Now let's integrate socket.io to this server. Include the following in the package.json to install socket.io in to the application.

```
{
  "name": "socket-chat-example",
  "version": "0.0.1",
  "description": "my first socket.io app",
  "dependencies": {}
}
```

Now just execute the following command in the terminal.

```
npm install --save
```

Ok, ow we have socket.io integrated to our application. Let's start using socket.io. Let's first create an index.html and serve it using node.js. This index.html will be the clients in our application. In the client let's first initialize to get the name of the user before starting the chat application.

```
<!DOCTYPE html>
<html>
    <head><title>Hello world</title></head>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io();
    </script>
    <body>
        <input type="text" name="name" value="" placeholder="Enter your
name!">
        <button type="button" name="button">Let me chat!</button>
    </body>
</html>
```

```
var app = require('express')();
var http = require('http').Server(app);

app.get('/', function(req, res){
  res.sendfile('index.html');
});

http.listen(3000, function(){
  console.log('listening on *:3000');
});
```

Here you can see that we have included the socket.io.js and created socket object in the index.html file. This is to include the client script and initialize the socket object there so that clients can establish connections when required. The script is served by our io server at **'/socket.io/socket.io.js'**. So basically when we create a client we need to include the socket.io.js file in order to integrate socket.io. In the server side code the only change that we have done is to return this index.html rather than sending a text in app.get method.

Since now we have created the initialization page for the chat application let's move our focus to the server. There are two main methods that we use in socket.io for real time web functionality, socket.on() and socket.emit(). Let's explore socket.on() method in the following server side code.

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendfile('index.html');
});

io.on('connection', function(socket){
  console.log('A user connected');

  socket.on('disconnect', function () {
    console.log('A user disconnected');
  });
});

http.listen(3000, function(){
  console.log('listening on localhost:3000');
});
```

In above code you can see that we have require the socket.io and created the io object. In this io object there is the method on which contains specific events that are fired. One such event in 'connection'. This event is raised when a socket.io client connects with the server. So in above code when the 'connection' event gets fired we have captured that event and tells to execute these functions for that specific socket after raising that event. 'disconnect' is also an event for that sepecific socket.io which raised when a client gets disconnected. Now if you run the application and open several tabs localhst:3000 and close you will see kind of following logs in your terminal.

```
A user connected
A user connected
A user disconnected
A user connected
A user disconnected
```

Now let's try to understand what is socket.emit() function. For that let's modify both the client and server for the purpose. Let's check when after entering the user name whether the user is already available and if not store user name and send it back to the client.

```
<!DOCTYPE html>
<html>
    <head><title>Hello world</title></head>
    <script src="/socket.io/socket.io.js"></script>
    <script>
```

```
      var socket = io();
        function setUsername(){
            socket.emit('setUsername', document.getElementById('name').
value);
        };
        var user;
        socket.on('checkUserExists', function(data){
            document.getElementById('error-container').innerHTML = data;
        });
        socket.on('userSet', function(data){
            user = data.username;
            document.body.innerHTML = '<input type="text" id="message">\
            <button type="button" name="button" onclick="sendMessage()"
>Send</button>\

<div id="message-container"></div>

        });
    </script>
    <body>
<div id="error-container"></div>
<input id="name" type="text" name="name" value="" placeholder="Enter
your name!">
        <button type="button" name="button" onclick="setUsername()">Let
me chat!</button>
    </body>
</html>
```

Here we can see we have used socket.emit() functionality to raise a method named 'checkUserExists'. So where is this 'checkUserExists' method?. It is on the server, so what socket.emit does is raise the method which it is calling on the other side. These events can be captured using socket.on() method like we have shown in previous code. To get a more clear idea let's implement the server side as well.

```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendfile('index.html');
});
users = [];
io.on('connection', function(socket){
  console.log('A user connected');
  socket.on('checkUserExists', function(data){
    console.log(data);
    if(users.indexOf(data) > -1){
      socket.emit('userExists', data + ' username is taken! Try some
other username.');
    }
```

```
    else{
      users.push(data);
      socket.emit('userSet', {username: data});
    }
  });
});
http.listen(3000, function(){
  console.log('listening on localhost:3000');
});
```

Now you can see that when the client emit the 'checkUserExists' method it is captured by socket.io in server side and the specified methods inside that function are executed. Here it will check for whether the user exists and send back the user data to the client using socket.emit() which will be captured by socekt.io() method in the client.

Now I think you have a pretty clear understanding on how socket.io transports data from client to server and server to client. All what remains in our application is to implement the functionality for chat. Let's think about what we need for that using what we have learn so far. Let's divide these methods in to two groups, client side methods and server side methods.

Client side methods

- We need to send message to the server. So we need an emit() method to send data to the server.
- We need to show chats from other users. These data are sent from the server, so we need an on() method to show these chats.

Let's try to implement these two requirements.

```
function sendMessage(){
            var msg = document.getElementById('message').value;
            if(msg){
                socket.emit('msg', {message: msg, user: user});
            }
        }
socket.on('newmsg', function(data){
            if(user){
                document.getElementById('message-container').innerHTML
+= '
<div><b>' + data.user + '</b>: ' + data.message + '</div>
}
})
```

Server side methods

- We need to get the chat data sent from clients. Hence we need an on() method to receive chat data.
- We need to send these chat data to other clients, hence we need an emit method()

Following are the socket.io implementation for above requirements.

```
socket.on('msg', function(data){
      //Send message to everyone
      io.sockets.emit('newmsg', data);
  })
```

That's it. We have implemented the chat application using socket.io. Following is the complete code for the client.

```
 <!DOCTYPE html>
<html>
    <head><title>Hello world</title></head>
    <script src="/socket.io/socket.io.js"></script>
    <script>
      var socket = io();
        function setUsername(){
            socket.emit('setUsername', document.getElementById('name').
value);
        };
        var user;
        socket.on('checkUserExists', function(data){
            document.getElementById('error-container').innerHTML = data;
        });
        socket.on('userSet', function(data){
            user = data.username;
            document.body.innerHTML = '<input type="text" id="message">\
            <button type="button" name="button" onclick="sendMessage()"
>Send</button>\

<div id="message-container"></div>

        });
        function sendMessage(){
            var msg = document.getElementById('message').value;
            if(msg){
                socket.emit('msg', {message: msg, user: user});
            }
        }
        socket.on('newmsg', function(data){
            if(user){
                document.getElementById('message-container').innerHTML
+= '


<div><b>' + data.user + '</b>: ' + data.message + '</div>


'
            }
        })
    </script>
    <body>
<div id="error-container"></div>
<input id="name" type="text" name="name" value="" placeholder="Enter
your name!">
```

```
        <button type="button" name="button" onclick="setUsername()">Let
me chat!</button>
    </body>
</html>
```

And following is the complete server side code.

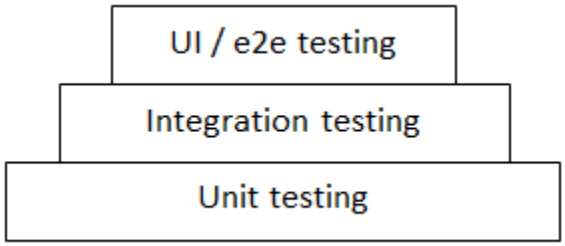
```
var app = require('express')();
var http = require('http').Server(app);
var io = require('socket.io')(http);

app.get('/', function(req, res){
  res.sendfile('index.html');
});
users = [];
io.on('connection', function(socket){
  console.log('A user connected');
  socket.on('setUsername', function(data){
    console.log(data);
    if(users.indexOf(data) > -1){
      socket.emit('userExists', data + ' username is taken! Try some
other username.');
    }
    else{
      users.push(data);
      socket.emit('userSet', {username: data});
    }
  });
  socket.on('msg', function(data){
      //Send message to everyone
      io.sockets.emit('newmsg', data);
  })
});
http.listen(3000, function(){
  console.log('listening on localhost:3000');
});
```

Now you can start the server and access 'localhost:3000' in different tabs to chat with several users using the real time web functionality.

Proof of Concepts

PoC - Integrating tests to Client Framework

Testing is required for an effective performance of software application or product because application failure can be very costly in the future. Test Driven Development makes code easier to maintain and produce automated test for regression testing. Automation testing is a crucial part of modern DevOps and angle approach enable to deliver higher quality software with fast and with confidence. Automation testing contains in three states.



Unit testing should be trying before the integration testing. Unit test check a single component of an application and should have no dependencies on code outside. Therefore I developed application and ran several unit testing and integration testing on the components prior to initiate the unit testing on Rex Application (client framework).

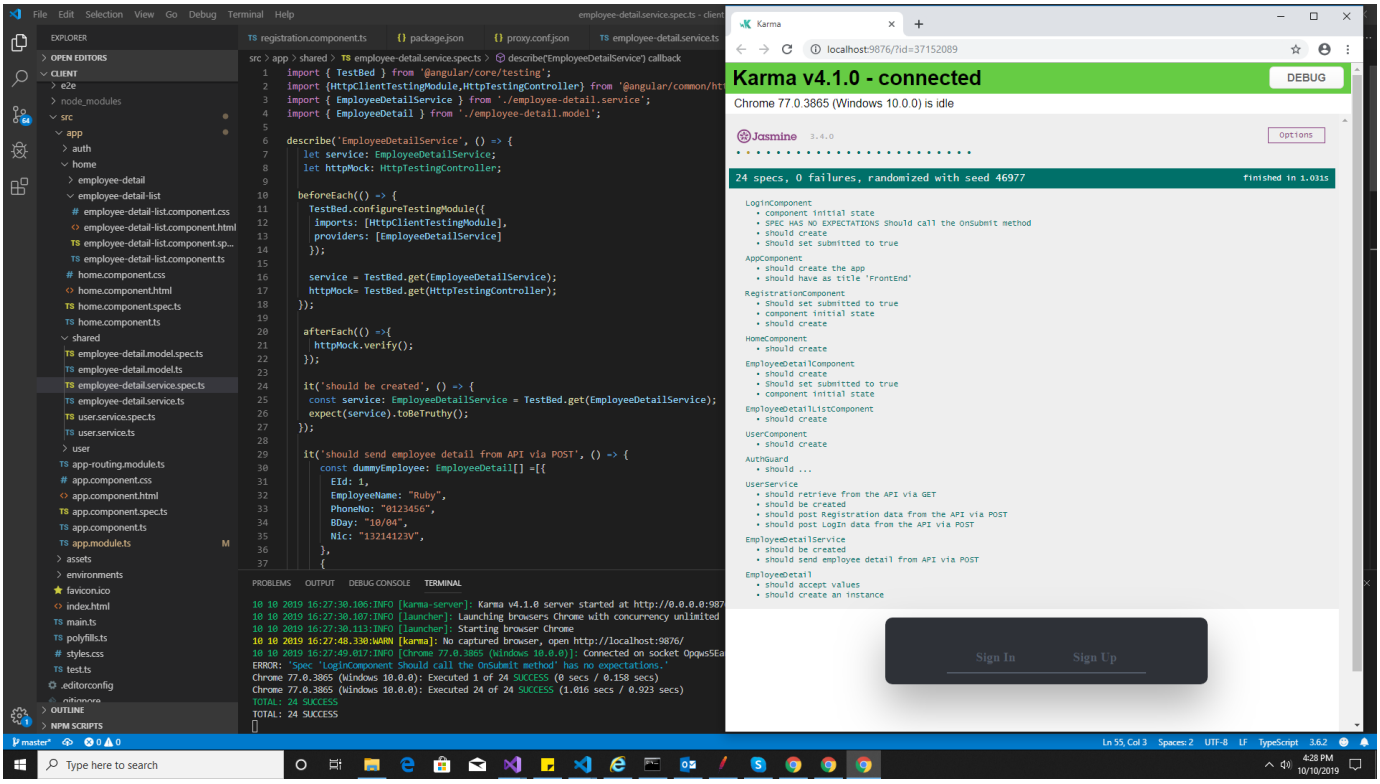
In POC application, client side is developed using angular framework. Thus I used karma and Jasmine in unit test. Karma is tool which monitoring for the changes of the develop file and return a test automatically. In simply karma is test runner for a relevant test. Jasmine is recommended testing framework in angular CLI.

Sample Unit test cases using karma and jesmine:

```
describe('EmployeeDetailService', () => {
    let service: EmployeeDetailService;
    let httpMock: HttpTestingController;
beforeEach(() => {
    TestBed.configureTestingModule({
    imports: [HttpClientTestingModule],
    providers: [EmployeeDetailService]
});
  service = TestBed.get(EmployeeDetailService);
  httpMock= TestBed.get(HttpTestingController);
});
afterEach(() =>{
    httpMock.verify();
});
it('should be created', () => {
   const service: EmployeeDetailService = TestBed.get(EmployeeDetailService);
   expect(service).toBeTruthy();
});
it('should send employee detail from API via POST', () => {
 const dummyEmployee: EmployeeDetail[] =
  [{
     EId: 1,
     EmployeeName: "Ruby",
     PhoneNo: "0123456",
     BDay: "10/04",
     Nic: "13214123V",
```

```
  },
  {
     EId: 1,
     EmployeeName: "Ruby",
     PhoneNo: "0123456",
     BDay: "10/04",
     Nic: "13214123V",
   },
];
  service.postEmployeeDetail().subscribe(employeeDetails =>{
   expect(employeeDetails.length).toBe(2);
   expect(employeeDetails).toEqual(dummyEmployee);
 });
   const request = httpMock.expectOne(`${service.rootURL}/EmployeeDetail`);
   expect(request.request.method).toBe('POST');
   request.flush(dummyEmployee);
});
it('should Delete employee detail from API via DELETE', () => {
const dummyEmployeeDelete: EmployeeDetail[] =
 [{
     EId: 1,
     EmployeeName: "Ruby",
     PhoneNo: "0123456",
     BDay: "10/04",
     Nic: "13214123V",
 },
 {
      EId: 1,
      EmployeeName: "Ruby",
      PhoneNo: "0123456",
      BDay: "10/04",
      Nic: "13214123V",
  },
];
    service.deleteEmployeeDetail(id).subscribe(employeeDetails =>{
        expect(employeeDetails.length-1).toBe(0);
        expect(employeeDetails).toEqual(dummyEmployeeDelete);
    });
      const requestDelete = httpMock.expectOne(`${service.rootURL}/EmployeeDetail/${id}`);
     expect(requestDelete.request.method).toBe('DELETE');
     requestDelete.flush(dummyEmployeeDelete);
```

```
});
it('should accept values', () => {
    let employeeDetail = new EmployeeDetail();
    employeeDetail = {
        EId: 1,
        EmployeeName: "Ruby",
        PhoneNo: "0123456",
        BDay: "10/04",
        Nic: "13214123V",
    }
  expect(employeeDetail.EId).toEqual(1);
  expect(employeeDetail.EmployeeName).toEqual("Ruby");
  expect(employeeDetail.PhoneNo).toEqual("0123456");
 expect(employeeDetail.BDay).toEqual("10/04");
 expect(employeeDetail.Nic).toEqual("13214123V");
});
```



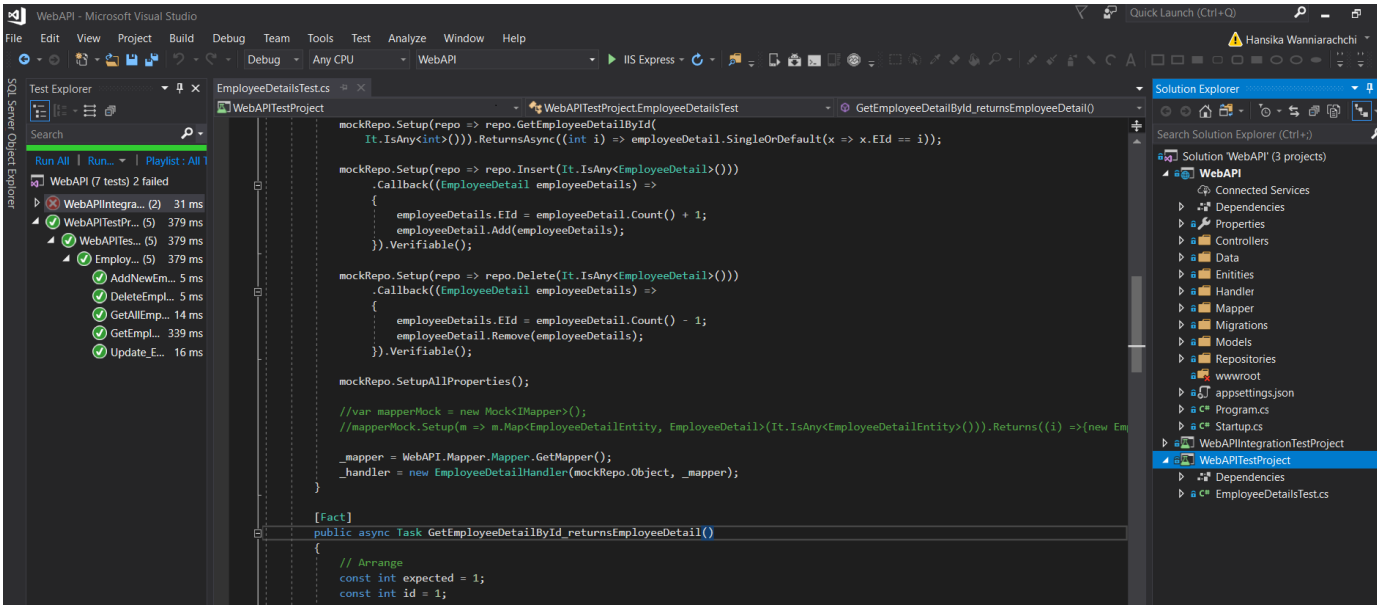
Server side of POC application is completed using asp.net core. Therefore XUnit used in unit testing because XUnit can implement its' own test in functionality with extensible attributes like [fact] and [Theory]. Further moq framework is used in unit testing that isolate the dependencies. moq is mocking framework for c# / ASP.NET which ensure that performance of unit testing is concise and quick.

Sample Unit test cases using Xunit:

```
public class EmployeeDetailsTest
 {
```

```
    private readonly EmployeeDetailHandler _handler;
    private readonly IMapper _mapper;

  public EmployeeDetailsTest()
  {
      var mockRepo = new Mock<IEmployeeDetailRepository>();

      IList<EmployeeDetail> employeeDetail = new List<EmployeeDetail>()
      {
         new EmployeeDetail
          {
              EId = 1,
              EmployeeName = "Hansika",
              PhoneNo = "0712656744",
              BDay ="01/12",
              Nic= "12345V"
          },
         new EmployeeDetail
          {
             EId = 2,
             EmployeeName = "Jayani",
             PhoneNo = "071211111",
             BDay ="01/01",
             Nic= "12345Y"
          } };

         mockRepo.Setup(repo => repo.GetAllAsync()).ReturnsAsync(employeeDetail.ToList());

         mockRepo.Setup(repo => repo.GetEmployeeDetailById(
             It.IsAny<int>())).ReturnsAsync((int i) => employeeDetail.SingleOrDefault(x => x.
EId == i));
              mockRepo.Setup(repo => repo.Insert(It.IsAny<EmployeeDetail>()))
             .Callback((EmployeeDetail employeeDetails) => {
                 employeeDetails.EId = employeeDetail.Count() + 1;
                 employeeDetail.Add(employeeDetails);
              }).Verifiable();

         mockRepo.Setup(repo => repo.Delete(
             It.IsAny<EmployeeDetail>())) .Callback((EmployeeDetail employeeDetails) => {
                employeeDetails.EId = employeeDetail.Count() - 1;
                employeeDetail.Remove(employeeDetails);
         }).Verifiable();

         mockRepo.SetupAllProperties();
         _mapper = WebAPI.Mapper.Mapper.GetMapper();
        _handler = new EmployeeDetailHandler(mockRepo.Object, _mapper);
  }

  [Fact]
  public async Task GetEmployeeDetailById_returnsEmployeeDetail()
  {
    // Arrange
    const int expected = 1;
    const int id = 1;

    // Act
    var employeeDetail = await _handler.GetEmployeeDetail(id);

    // Assert
    Assert.Equal(expected, employeeDetail.EId);
  }

  [Fact]
  public async Task GetAllEmployees_returnsEmployeeDetails()
  {
    // Arrange
    const int expected = 2;

    // Act
    var employeeDetails = await _handler.GetEmployeeDetails();

    // Assert
```

```
  Assert.Equal(expected, employeeDetails.Count());
}

[Fact]
public async Task AddNewEmployee()
{
 // Arrange
 var EmployeeDetail = new EmployeeDetail()
  {
    EId = 1,
    EmployeeName = "Hansika",
    PhoneNo = "0712656744",
    BDay = "01/12",
    Nic = "12345V"
   };

 // Act
 var employeeDetail = await _handler.PostEmployeeDetail(EmployeeDetail);

// Assert
Assert.Equal(EmployeeDetail.EId, employeeDetail);
}

[Fact] public async Task DeleteEmployee()
{
  // Arrange
  var EmployeeDetail = new EmployeeDetail
   {
      EId = 1,
      EmployeeName = "Hansika",
      PhoneNo = "0712656744",
      BDay = "01/12",
      Nic = "12345V"
   };

 // Act
 var employeeDetail = await _handler.DeleteEmployeeDetail(EmployeeDetail.EId);

 // Assert
 Assert.Equal(EmployeeDetail.EId, employeeDetail);
}
```



```
```

After testing all the possible test cases by unit testing we can move on to integration testing because integration testing sits in middle of the automation test pyramid. Unit testing generally intended to test a specific component in an isolated test environment but a software does not tend to operate in isolated environment it interacts with third party components such as web servers and databases. Therefore, integration testing takes place. Xunit is used to perform integration testing for server side application.

Sample Integration Test cases:

```
class TestClientProvider
   {
      public HttpClient Client { get; private set; }
      private readonly TestServer _server;

      public TestClientProvider()
       {
            var projectDir = System.IO.Directory.GetCurrentDirectory();
            _server = new TestServer(new WebHostBuilder()
            .UseEnvironment("Development")
            .UseContentRoot(projectDir)
            .UseConfiguration(new ConfigurationBuilder()
            .SetBasePath(projectDir)
            .AddJsonFile("appsettings.json")
            .Build()
            )
           .UseStartup<Startup>());

        Client = _server.CreateClient();
        Client.DefaultRequestHeaders.Accept.Add(
          new MediaTypeWithQualityHeaderValue("application/json"));
       }
    }

   public class EmployeeIntegrationTest
    {
       [Fact]
       public async Task Test_Get_All()
        {
          using (var client = new TestClientProvider().Client)
          {
            var response = await client.GetAsync("/api/EmployeeDetail");
            Assert.Equal(HttpStatusCode.OK, response.StatusCode);
           }
         }
       }
```

As mention above, after successfully passing all the unit testing and integration testing cases. Our next milestone is to initiate integration testing for client framework.

Hatteland Way

Please go through this document and follow this if you are a new member to the team

- Attendence patterns
- DevOps
- Domain knowledge Grooming
- Dresscode
- Mentoring
- Preparing for workshops
- Skill building
- Team code of Ethics
- Team communication
- Value addition to customer
- Working from Home
- New Recruitment On Boarding
- Journey map
- POCs
- Cloud computing

Attendence patterns

Daily Attendance

Scrum

Demo

Individual Skype meetings

Technical grooming meetings

Business grooming meetings

Responding to skype messages

Approvals before taking leave

Avoid last minute notification

Late arrivals to office

Frequent attendance to external meetings / Events (SC ...)

https://seranet.atlassian.net/wiki/display/FAHR/HRIS+Video+Tutorials

https://seranet.atlassian.net/wiki/display/FAHR/Attendance

DevOps

PoC - CI/CD

1. Clone
2. Build
3. SonarQube
4. Run unit tests
5. Run integration tests
6. Deploy - Staging and Production

If any of the stages fail, send an email.

If all the stages are successful, send an email.

Technologies

Jenkins

SonarQube

Azure - CI/CD

Task	Status

Create a Linux virtual machine in Azure	Done
Install Docker in VM	Done
Jenkins in Docker	Done
Export local jobs to Jenkins	Done
Configure the tokens & keys in Jenkins	Done
Run SonarQube container	Done
Install IIS in a separate VM and update the pipeline stage	_
Create Azure SQL Database	Done
Deploy .net core & angular projects	
Finalized the e-mail notification content	Done

Repository: https://github.com/HansikaW/hatteland-poc

Setup Jenkins Docker image:

1. Pull jenkins image

```
docker pull jenkins/jenkins:lts
```

2. Use jenkins image

```
docker run -u 0 -v /var/run/docker.sock:/var/run/docker.sock -v
$(which docker):$(which docker) -p 8080:8080 -v /usr/var
/jenkins_home jenkins/jenkins:lts
```

3. Create user group docker within jenkins docker container

```
groupadd docker
```

4. Add user Jenkins to user group ‘docker’

```
useradd -G docker Jenkins
```

5. Let’s open localhost:8080 (or whatever you have chosen as the host port).Grab administrative password from the Docker logs and paste it, Choose the suggested plugins option and Jenkins will start the installation of plugins.

SonarQube with Jenkins Setup using Docker Image

1. Install SonarQube Scanner Jenkins plugin (Manage Jenkins > Manage Plugins > Available)
2. Download the official SonarQube image from Docker Hub

```
docker pull sonarqube
```

3. Start the server

```
docker run -d --name sonarqube -p 9000:9000 sonarqube
```

4. Starts SonarQube on port 9000 and login to SonarQube with the default admin user and admin password
5. SonarQube Scanner Configuration

   i. access the Jenkins Docker container from a bash shell like this:

```
docker exec -it {Jenkins-CONTAINER ID } /bin/bash
```

   ii. create sonar-scanner directory under **both /var/jenkins_home and root directory of your jenkins Container**

   iii. download SonarQube Scanner onto the container from the sonar-scanner directory with wget:

```
 wget https://binaries.sonarsource.com/Distribution/sonar-scanner-
cli/sonar-scanner-cli-3.3.0.1492-linux.zip
```

   iv. unzip the Sonar Scanner binary:

```
 unzip sonar-scanner-cli-3.3.0.1492-linux.zip
```

   v. in Jenkins: update Jenkins to point to sonar-scanner binary (Manage Jenkins > Global Tool Configuration > SonarQube Scanner); you will need to uncheck “Install automatically” so you can explicitly set SONAR_RUNNER_HOME

Configuring Jenkins and SonarQube

1. In SonarQube: add webhook in SonarQube to point to Jenkins (Administration > Configuration > Webhooks); URL will be in the format http://<host_ip>:8080/sonarqube-webhook
2. In SonarQube, generate an access token that will be used by Jenkins (My Account > Security > Tokens)
3. In Jenkins, add the SonarQube Server IP address and the access token` (Manage Jenkins > Configure System > SonarQube Servers); URL will be in the format http://<host_ip>:9000
   - SonarQube access token can be found in sonarQube: MyAccout(top right side corner) > security --> Token
4. Create a sonarQube project or give jenkins pipeline script as follows:

```
bat 'dotnet sonarscanner begin /k:Hatteland-POC /d:sonar.host.url="
http://<host-ip>:9000" /d:sonar.login=admin /d:sonar.password=admin'
    bat "dotnet build <project folder> --configuration Release"
    bat  "dotnet sonarscanner end /d:sonar.login=admin /d:sonar.
password=admin"
```

Configuring Jenkins and Github

1. In jenkins add GitHUB server,(Manage Jenkins > Configure System > GitHub Server)

   set API URL as https://api.github.com

   for credential: use secret text taken from github (github settings > Developer settings > personal Access token)
2. Add github-webhook for relevant github repository (git-repository > Webhooks > Add webhook)

Payload URl : http:///github-webhook

```
Content type: application.json
```

Database connection

Add connections from the your specified IPs that will provides access to the databases.

Install

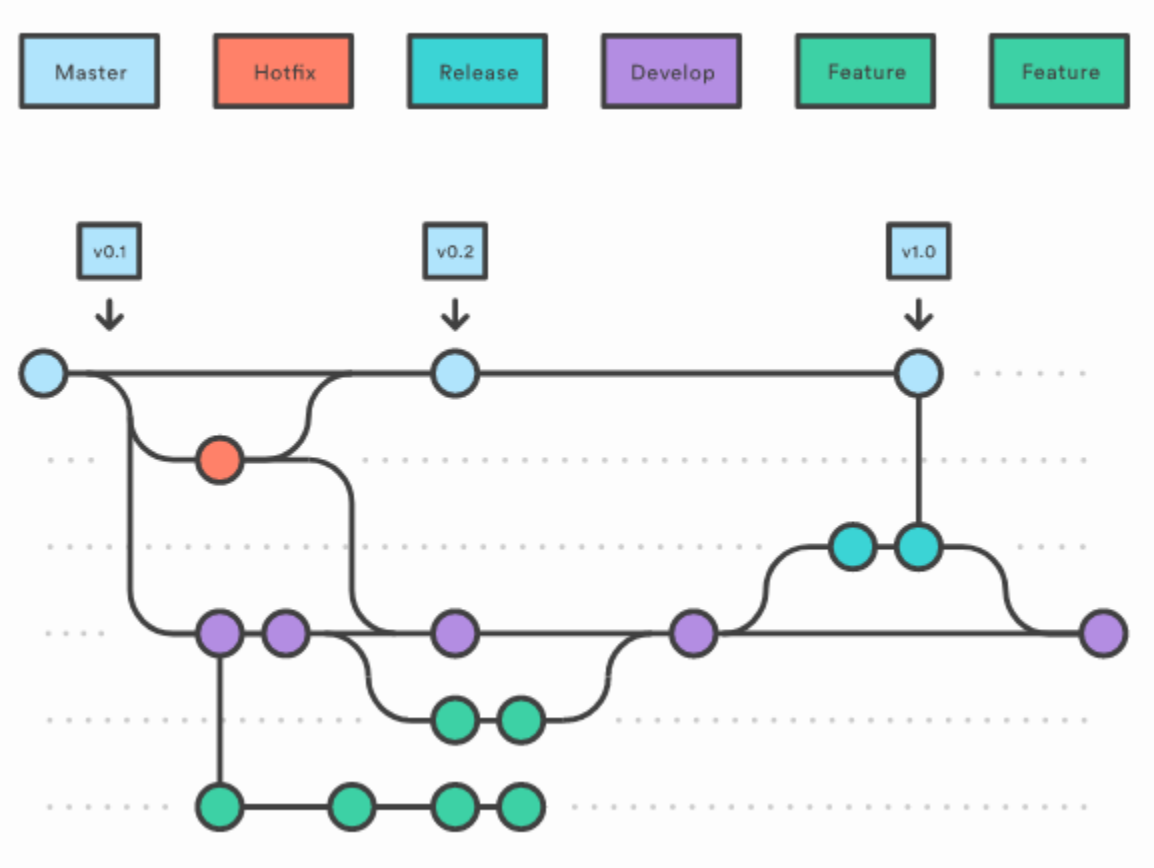
- Install the .NET Core SDK in jenkins container
- Install docker in Jenkins container
- Install node and angular cli

Jenkins pipeline

As this is single branch repository, select pipeline as item and give pipeline name. (if your repository has multiple branches then you can move to the multibranch pipeline) After that mark the Github project and add your repository URL. Under the Buil Trigger, mark the GitHub hook tigger for GITScm polling as a option.

- should create two seperate pipelines for the client application(Angular) and the server project(.Net core)
- sperate jenkins pipelins scripts are attached in this repostory(jenkinsfile-client & jenkinsfile-server).

Git flow and Azure Pipeline



- Master - Main branch with stable code
- Develop - Branch for the staging environment
- Release - Release branch for individual releases which will be released (branched off from Develop)
- Feature - Individual feature development
- Hotfix - Spontaneous fixes

Development flow

- Feature development will begin with the branching off from Develop branch
- Once the feature branch is developed and pushed to Bitbucket, Azure DevOps will run ONLY the build stage. (specified using conditions in stages)
- A PR is created to merge the feature with Develop branch
- A build will be run Azure DevOps for the PR (ONLY build stage no deployment still)
- After the PR is merged, Azure DevOps will run the build and deploy to the staging environment
- After a few features are developed, we come to release where we branch off from Develop to a release/* branch/
- Azure DevOps will run a build for release/* branch
- Merge release with Develop.
- Azure DevOps will first run-up to deploying to the staging environment for Develop branch merge
- Merge release with Master
- Azure DevOps will first run-up to deploying to the production environment for Master branch merge
- Hotfix
  - Create a branch from the master branch with hotfix/* naming convention
  - Do the changes
  - Merge back to master branch
  - Azure DevOps will run up to production deployment (this can be configured in Azure DevOps to skip staging deployment and directly promote a build to production)

Domain knowledge Grooming

Participation

Preparation

Retro

Build ERP expertise

Understand the user and support center operations

UI team to learn Javascript

UI team to learn Domain knowledge

Dresscode

Work dress code

Onsite dress code

What to dress and what not to dress

Definition of what is smart dress means

How to cut hair

99X Technology Dress Code

Mentoring

Mentoring Program

Preparing for workshops

How to know customer expectation ?

What are our expectations ?

What is customer schedule ?

What is our schedule ?

Skill building

1. Technical skill building
2. Communication skill building

Talent Bandwidth

Team code of Ethics

Roles & Responsibilities

Team communication

Value addition to customer

With the experience of over 150 project we must have very good skills in following areas and we should give benefits to customer always

User Experience

DevOps

Security

Process

Working from Home

Work From Home Policy

New Recruitment On Boarding

Journey map

RPS

Source	Personal Data	Reason	Handling	Disposal
Print connector registration form	PID Password Cookie	Authenticate the user to the system	Rambase on-premise database	Upon request
Rambase API	Email	To send Print connector registration details to the user when creating a connector for the first time	Rambase in-memory	Removed from memory after sending the mail
Rambase API	Output files (eg: invoices)	To print from Print cloud and connector applications	Rambase on-premise file server	Deleted after printing

RCF

Applications	Fields	Reason	Handling	Disposal
General	PID Email Password	Authenticate the user to the system	Authenticate the user to the system with appropriate encryption.	Upon request
CONTACT	Source	Source of which the personal information was obtained	Rambase on-premise database	Upon request

CONTACT	Legal basis	The legal basis for processing contact's data	Rambase on-premise database	Upon request
CONTACT	Privacy policy accepted	Accepts privacy policy	Rambase on-premise database	Upon request
CONTACT	Consent to marketing	Accepts marketing consent	Rambase on-premise database	Upon request

Reference
https://rambase.atlassian.net/browse/RC-567

POCs

HTTP2 Load Test

Http2

History of HTTP

The Hypertext Transfer Protocol (HTTP), the simple, constrain application layer protocol forms the foundation of the World Wide Web that is used for transferring information throughout. This protocol defines how messages are formatted and transmitted, and what actions web servers and browsers should take in various commands. The first documented version of HTTP was released in 1991 as HTTP0.9, which later led to the official introduction and recognition of HTTP1.0 in 1996 followed by HTTP1.1 in 1997 and has since received little iterative improvements. After that in February 2015, the Internet Engineering Task Force(IETF) developed the second major version of the application protocol in the form of HTTP/2. Most recently, In September 2019 IETF introduced HTTP/3 as the third major version of the Hypertext Transfer Protocol used to exchange binary information on the World Wide Web, succeeding HTTP/2.

Introduction to HTTP/2

HTTP/2 makes applications faster, simpler and more robust with many of the HTTP/1.1 workarounds. The primary goals of HTTP/2 are to reduce latency by enabling full request and response multiplexing, minimize protocol overhead through efficient compression of HTTP header fields, and add support for request prioritization and server push. HTTP/2 does not modify the application semantics of HTTP in any way. All the core concepts, such as HTTP methods, status codes, URLs and header fields, remain in place which means all existing applications can be delivered without any modification.

Features of HTTP/2

Binary Framing Layer

HTTP/2 introduces a new binary framing layer, which dictates how the HTTP messages are encapsulated and transferred between the client and server by transforming from a text protocol to a binary protocol. This protocol uses binary commands (in 1s and 0s) to execute the complete request-response cycles. Browsers using HTTP/2 implementation will convert the text commands into binary before transmitting it over the network. This feature will enable key business advantages for internet companies and online business as detailed with benefits of HTTP/2 such as Low overhead for parsing data, Light network footprint, Efficient and robust in terms of processing of data between client and server, and Reduced network latency and improved throughput

Request and Response Multiplexing

With HTTP/1.1 only one response can be delivered at a time (response queuing) per connection. With HTTP/2, new binary framing layer removes these limitations, and enables full request and response multiplexing, by allowing the client and server to break down an HTTP message into independent frames, interleave them, and then reassemble them on the other end. This approach presents an array of benefits, like the parallel multiplexed requests and responses which do not block each other and no need to apply unnecessary optimization hacks, such as image sprites, concatenation, and domain sharing among others that compromise other areas of network performance.

Stateful Header Compression

Each HTTP transfer carries a set of headers that describe the transferred resource and its properties. For delivering high-end web user experience requires websites rich in content and graphics. HTTP/2 implementation addresses these concerns with the ability to compress a large number of redundant header frames. It uses the HPACK specification as a simple and secure approach to header compression. Both client and server maintain a list of headers used in previous client-server requests. This feature provides effective stream prioritization, effective utilization of multiplexing mechanism and reduces resource overhead.

HTTP/2 Server Push

Another powerful new feature of HTTP/2 is the ability of the server to send multiple responses for a single client request. That is, in addition to the response to the original request, the server can push additional resources to the client without the client having to request each one explicitly. Then we can eliminate the extra latency and let the server push the associated resources ahead of time. The client saves pushed resources in

the cache and the client can reuse these cached resources across different pages. The server can multiplex pushed resources along with originally requested information within the same TCP connection. A key performance differentiator in HTTP/2 vs HTTP1 was that the server can prioritize pushed resources.

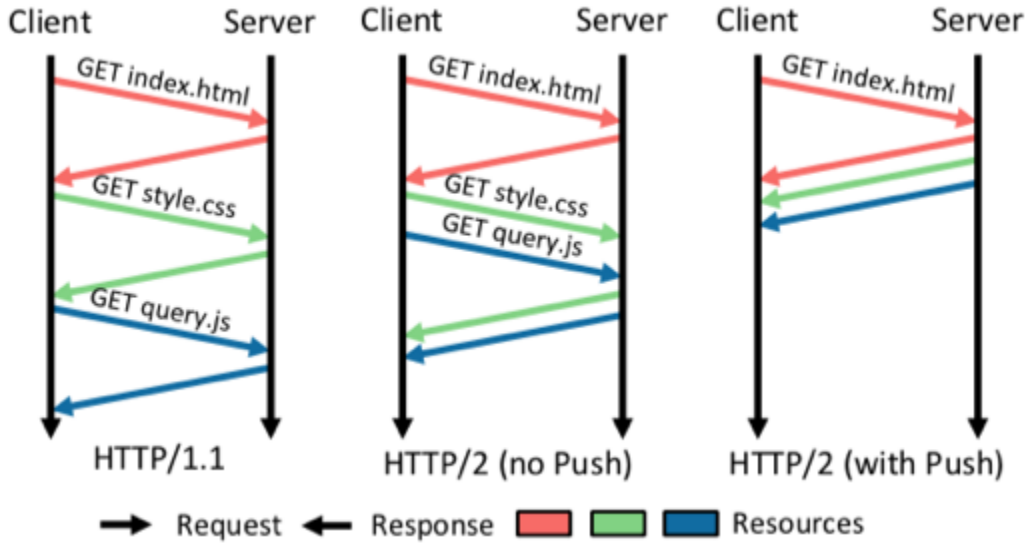


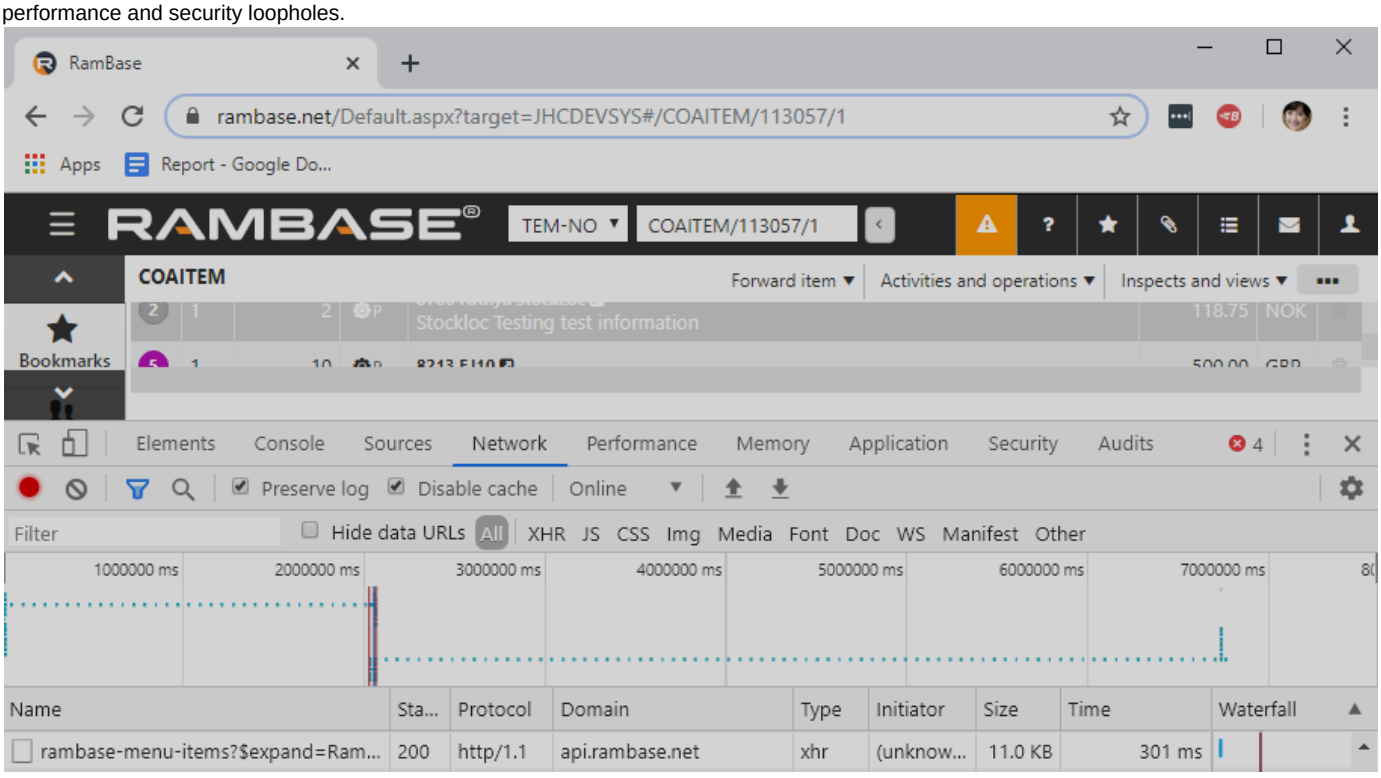
Figure 1 — HTTP/2 compared to HTTP/1.1 and HTTP/2 without Server Push

PUSH PROMISE - All server push streams are initiated via PUSH-PROMISE frames, which signal the server's intent to push the described resources to the client and need to be delivered ahead of the response data that requests the pushed resources. The simplest strategy to satisfy this requirement is to send all PUSH-PROMISE frames, which contain just the HTTP headers of the promised resource, ahead of the parent's response.

What was wrong with HTTP/1.1

HTTP1.1 was limited to processing only one outstanding request per TCP connection, forcing browsers to use multiple TCP connections to process multiple requests simultaneously. However, using too many TCP connections in parallel leads to TCP congestion that causes unfair monopolization of network resources. Web browsers using multiple connections to process additional requests occupy a greater share of the available network resources, hence downgrading network performance for other users.

Ineffective use of the underlying TCP connections with HTTP1.1 also leads to poor resource prioritization, causing exponential performance degradation as web applications grow in terms of complexity, functionality, and scope. The web has evolved well beyond the capacity of legacy HTTP-based networking technologies. The core qualities of HTTP1.1 developed over a decade ago have open to several embarrassing performance and security loopholes.



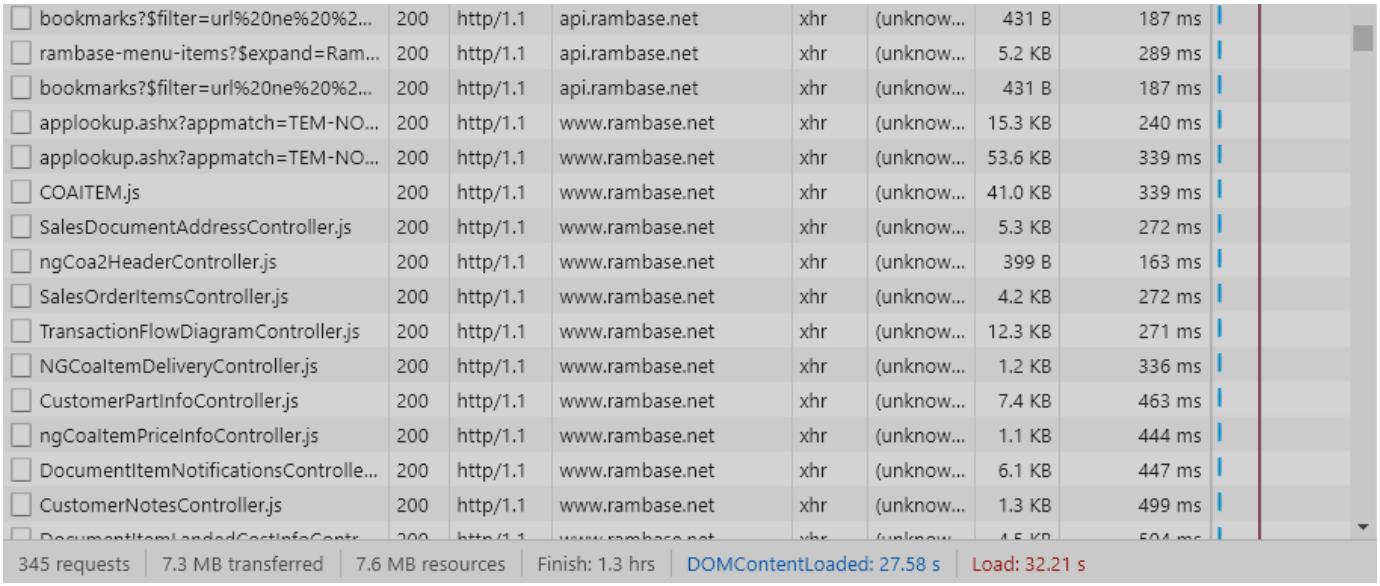


Figure 2 — Rambase.net with HTTP/1.1 Protocol

HTTP/2 on ASP.NET

Requirement for HTTP/2

- HTTP/2 support was introduced in IIS 10.0
- HTTP/2 was not supported IIS 8.5 and earlier.
- IIS running on Windows 10 or Windows Server 2016 supports HTTP/2
- HTTP/2 supports for .net framework 4.6 or later with PushPromise API
- HTTP/2 needs to be supported by both the server and the client (If either party doesn't support HTTP/2, both will use HTTP/1.1.)

Note: ASP.NET Core 2.x does not have support for HTTP/2 push promises right now. Therefore, the Server Push feature is not in the ASP.NET Core 2.x. The direct HTTP/2 support for Kestrel is still in backlog as it this blocked due to its missing ALPN support in SSL Stream as for alternative solution application can be hosted on HttpSysServer. This to the ASP.NET Core 3.0 milestone and moreover ASP.NET 3.0 provides the facility to enable HTTP/2 in two ways: at the HttpClient instance level or the HTTP request levels.

Step to Set-Up HTTP/2

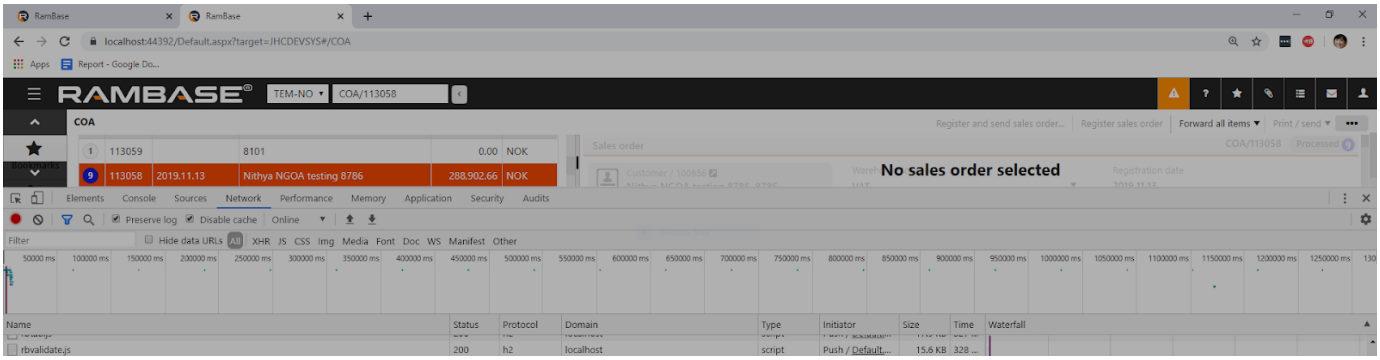
- Ensure HTTPS is enabled: Purchased an SSL or TLS certificate from a valid issuing authority, or utilize free SSL
- Activate the security certificate
- Install the certificate
- Update the website to enable the HTTPS protocol - IIS configuration settings specific to HTTP/2
- Launch your browser from your Windows 10 or Windows Server 2016 machine and hit F12

HTTP/2 Server Push on ASP.NET MVC

The IIS built-in support for Server Push is exposed to the ASP.NET through HttpResponse. PushPromise method takes a virtual path to the resource which is supposed to be pushed. Therefore, whenever a link element such as scripts, styles, and images are being rendered the helper registers the push as well.

HTTP/2 on Rambase

After enabling HTTPS and IIS configuration, Rambase will be work on relevant HTTPS port locally with HTTP/2. It can be seen that the protocol column in the network tab is h2 (HTTP/2) in the below image.



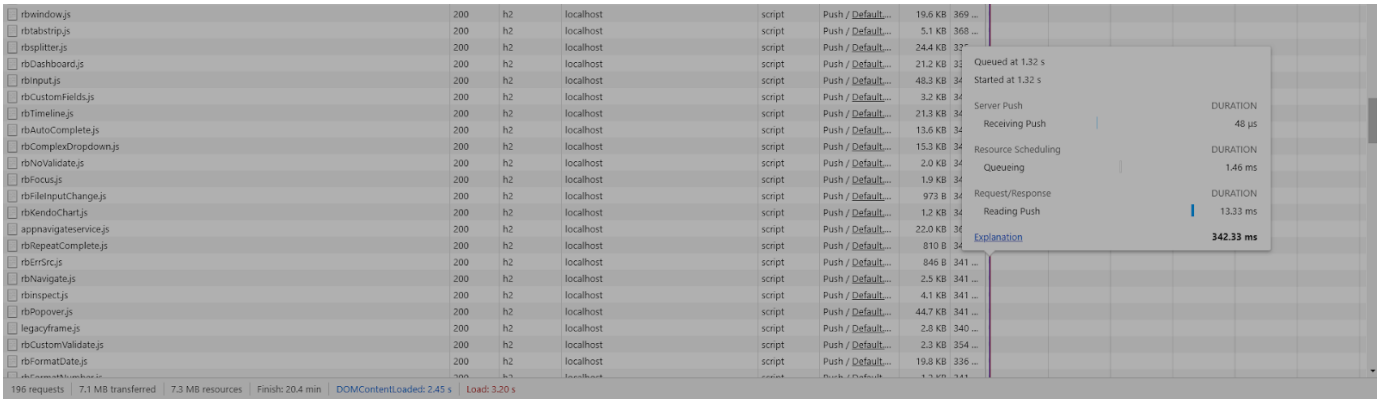


Figure 3 — HTTP/2 with ServerPush Features on Rambase

HTTP/2 ServerPush on Rambase

PUSH_PROMISE method used to apply the HTTP/2 server push because the server creates the PUSH_PROMISE frame to the response part of a normal browser-initiated stream. Response objects with the context of a reques2t which has HTTP connection are used to server push. In Rambase under the Page_load method which has HTTP connection can be used to apply Response. PUSHPROMISE for a push all the relevant scripts, styles and images without the client having to request each one explicitly. The Initiator column on network Tab in the hosted page shows that the request has been PUSH via HTTP/2 as figure 3.

Bottlenecks in Rambase

- Some common API requests are going through IIS 8.5 therefore, automatically request will use HTTP/1.1

HTTP/2 Performance Testing

In the context of web development, performance testing entails using software tools to simulate how an application runs under specific circumstances. Quantitative performance testing looks at metrics like response time while qualitative testing is concerned with scalability, stability, and interoperability. Performance Testing means testing a web application against heavy load, multiple and concurrent user traffic. Although HTTP/2 testing tools are helpful and can give a quick overview of speed and overall performance, also can conduct analysis using a variety of tests such as load tests, stress test, etc. For Rambase HTTP/2 performance testing both load tests and stress tests were carried out.

In simple terms, Load tests look at how increased workload affects an application's response time. For example, load testing tools use to see how the application performs with a certain number of simultaneous users. The purpose of load testing is to evaluate how the application behaves under normal working conditions. On the other hand, Stress Tests are also similar to load tests, but they look at how an application performs outside the boundaries of normal working conditions. The goal of stress testing is to determine how many concurrent users or transactions the application can handle before it crashes. As performance testing tool Apache Jmeter was used to carry out Load and stress tests.

What is Jmeter?

Apache Jmeter is a software tool that will simulate real-user behaviors and performance/load test of the site. Apache JMeter uses to analyze and measure the performance of web applications or a variety of service and also Jmeter support for multi-protocols, full multithreading framework and visualize test results.

Jmeter Workflow

JMeter simulates a group of users sending requests to a target server, and return statistics information of the target server through generating a test report in a different form. Jmeter HTTP/2 Plugin used to test the HTTP/2 Protocol. That complete workflow of Jmeter as shown in the figure below.

Figure 4 —Jmeter Workflow Diagram

Load Test Analysis:

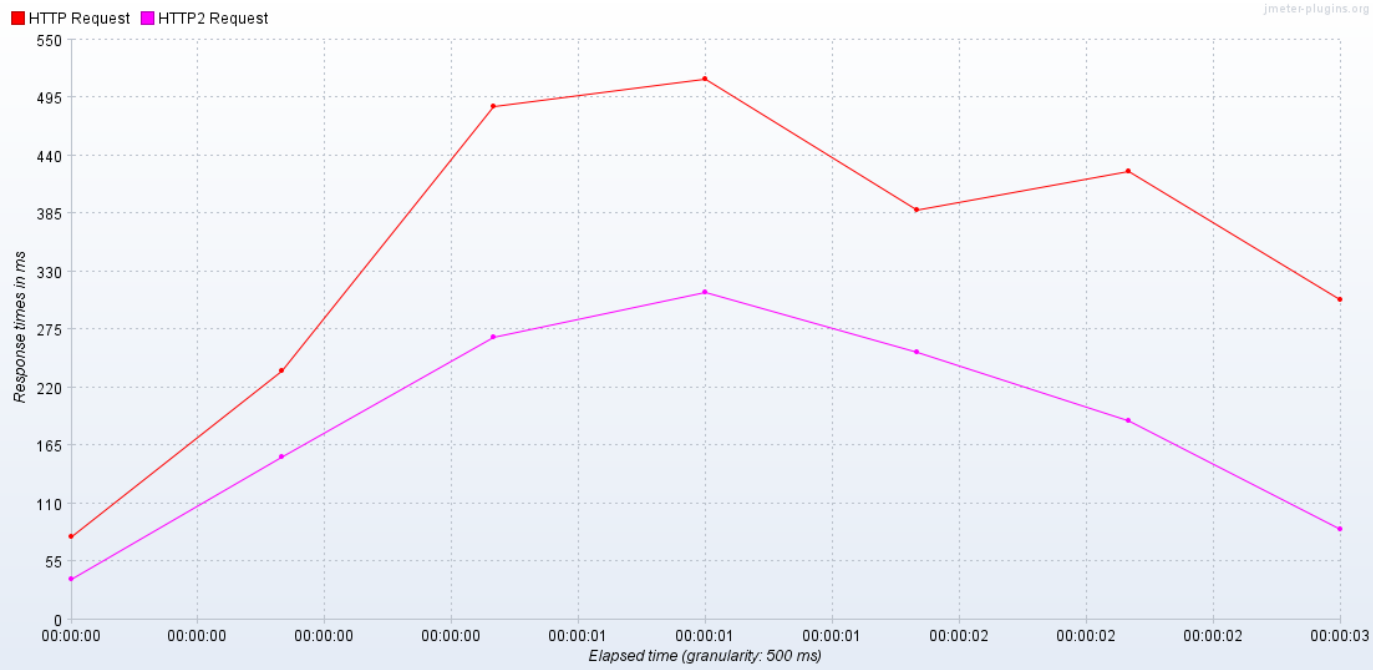
Response Times Over Time

Response time is the difference between the time when the request is sent and the time when the response has been fully received. Response Time is also known as Load time. This analysis compared the response time of HTTP/2 vs HTTP/1.1 with various amounts of users, each result came under load testing because this shows how Rambase performs with a certain number of simultaneous users with HTTP/1.1 protocol (D) and HTTP/2 protocol(d). The following graph will display for each sampler the average response time in milliseconds.

(1) Number of threads(Users): 20

Ramp-up time: 1 sec

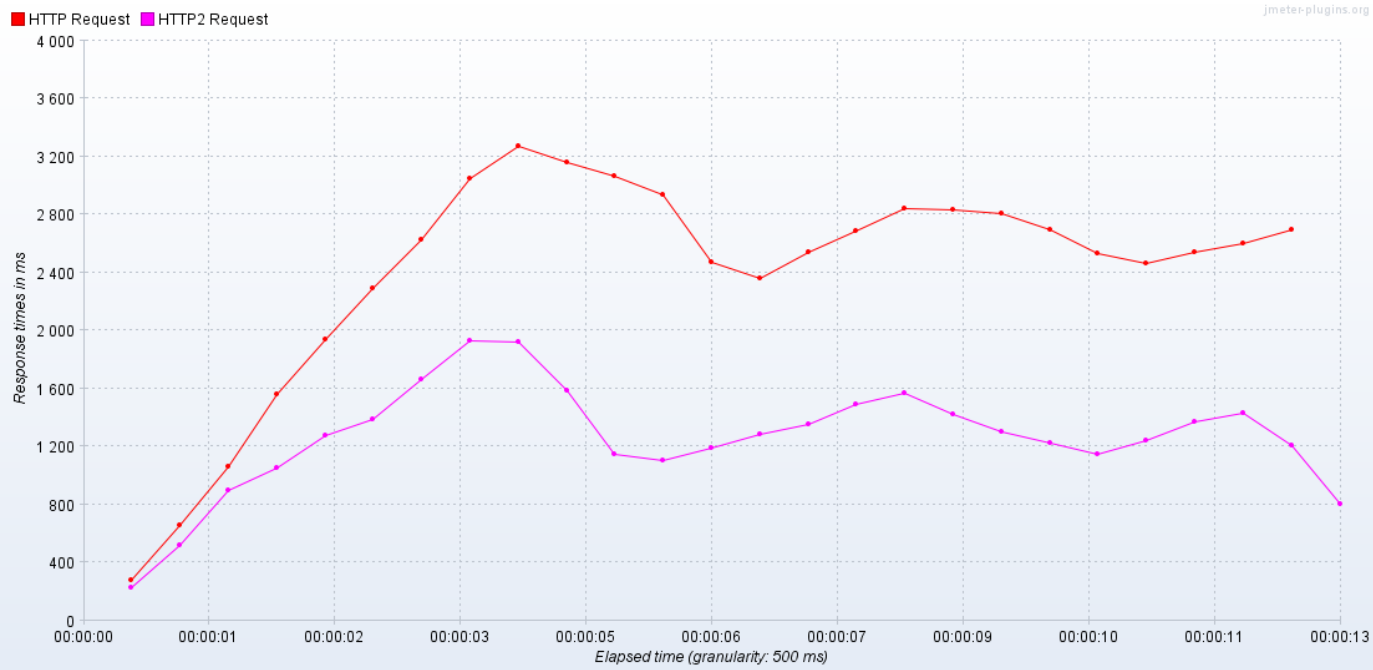
Number of Loops: 3



(2) Number of threads(Users) : 100

Ramp-up time: 1 sec

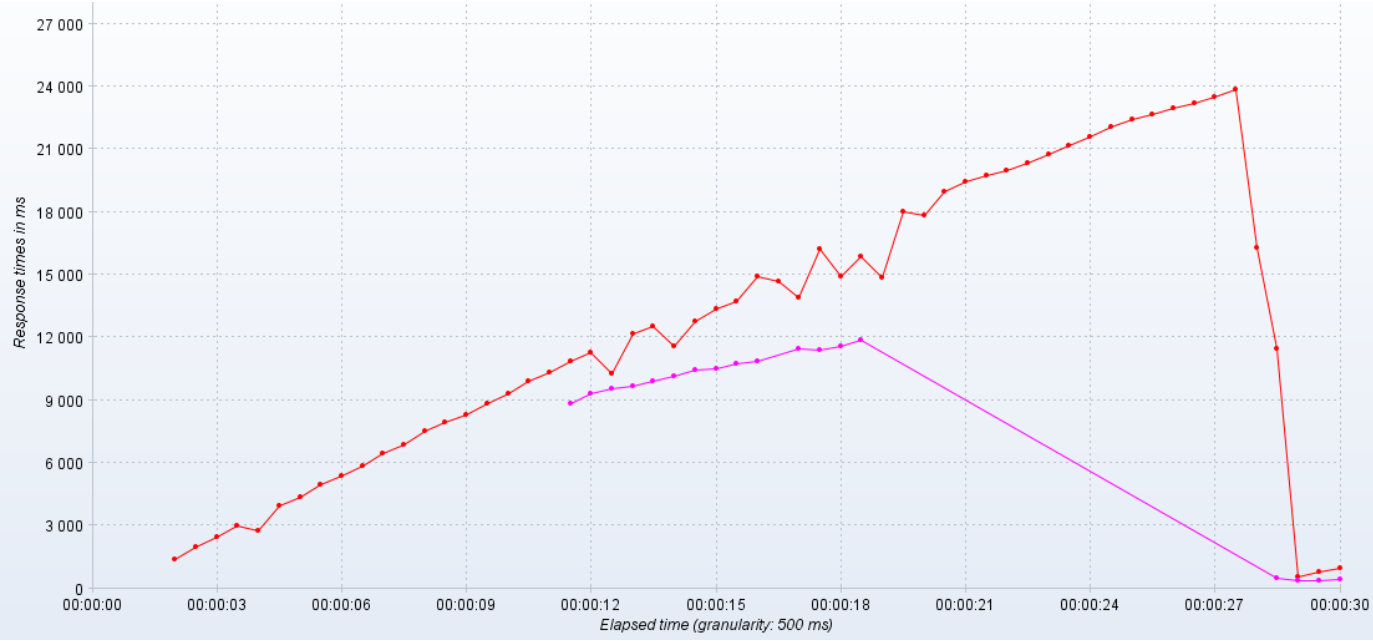
Number of Loops: 3



(3) Number of threads(Users) : 800

Ramp-up time: 1 sec

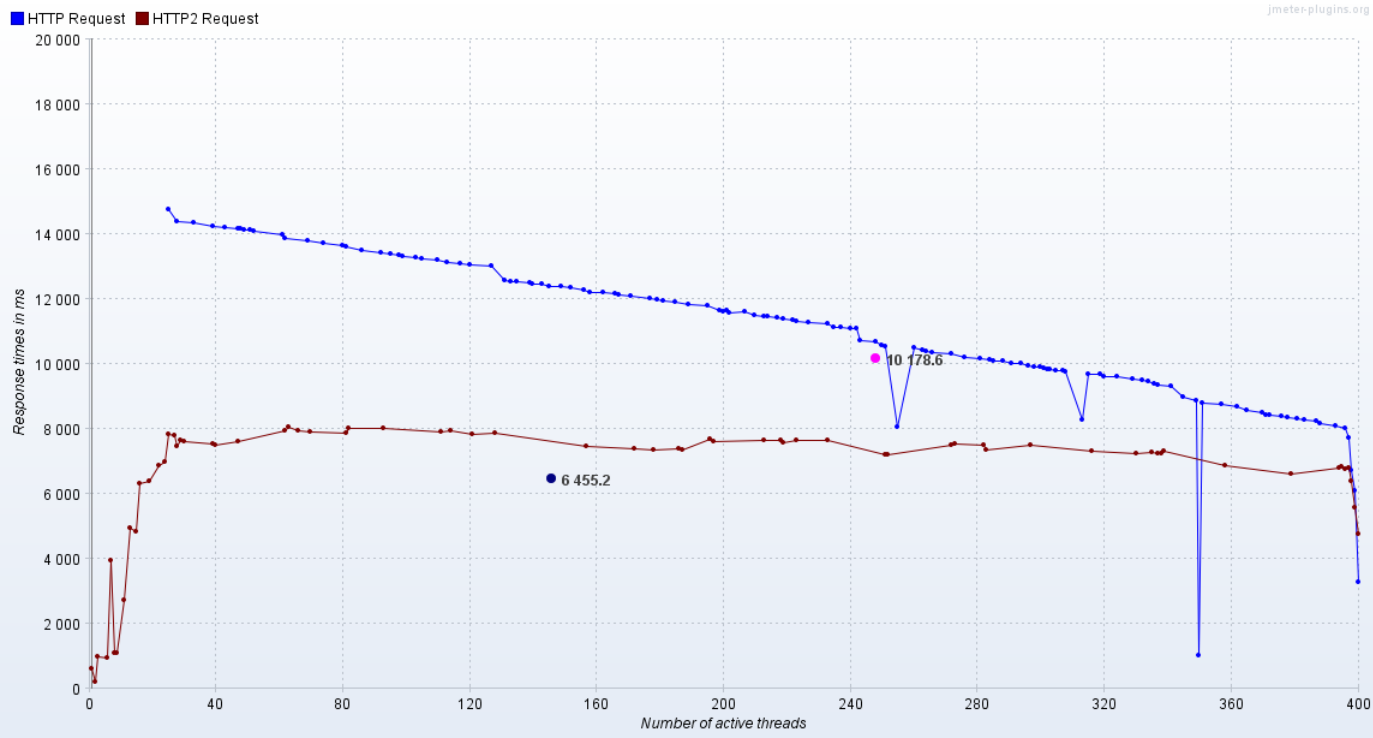
Number of Loops: 3



According to the results shown in the above graphs, HTTP/2 has less response time than the HTTP/1.1 with any number of threads. When the number of threads is high, HTTP/2 takes more time to start the response [Elapsed time] for instance it can be seen when the number of threads is 800, the first response time of HTTP/2 is 00.00.10 millisecond.

Response Time vs Thread

This analysis shows how response time changes with the number of parallel threads. Naturally, the server takes longer to respond when a lot of users request it simultaneously. The below graph visualizes such dependencies.



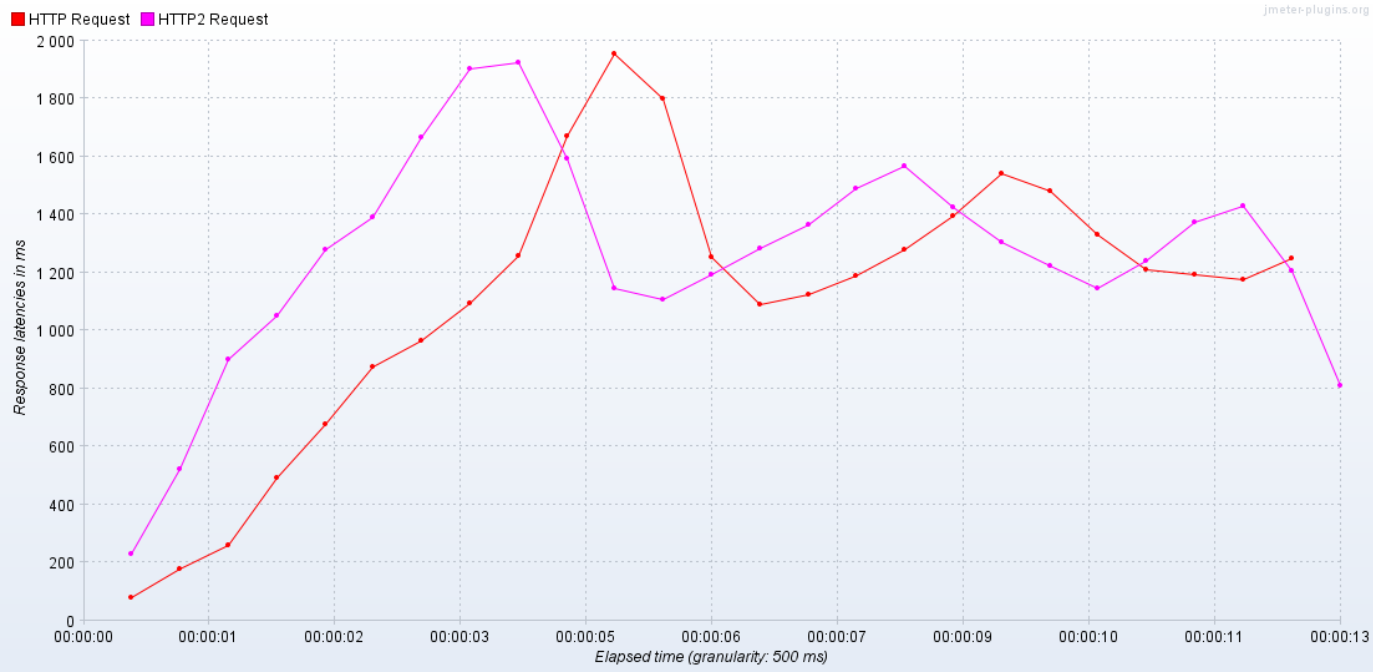
The above graph clearly shows that the response time of HTTP/2 protocol has stable behavior over the number of active threads and also response time of HTTP/2 protocol is relatively less with the average response time of 6455.2 ms. where HTTP/1.1 response time is 10178.6 ms

Response Latency Over Time

(1) Number of threads(Users) : 20

Ramp-up time: 1 sec

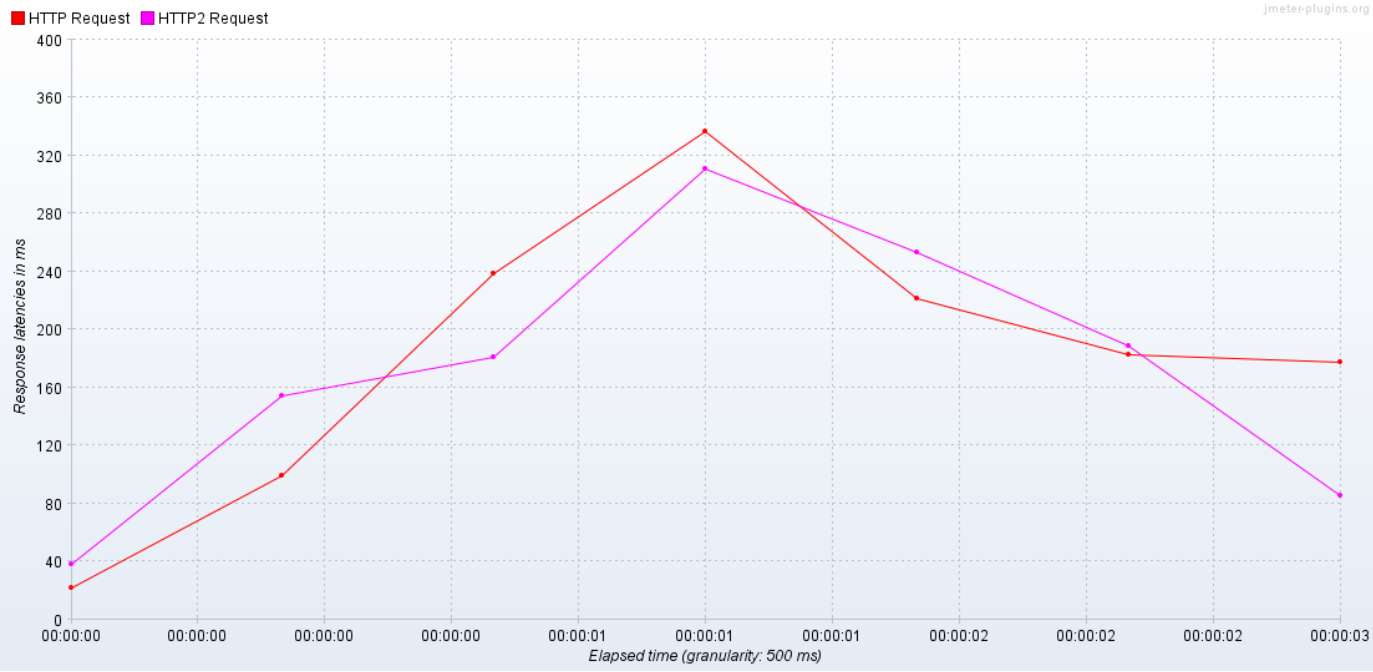
Number of Loops: 3



(2) Number of threads(Users): 100

Ramp-up time: 1 sec

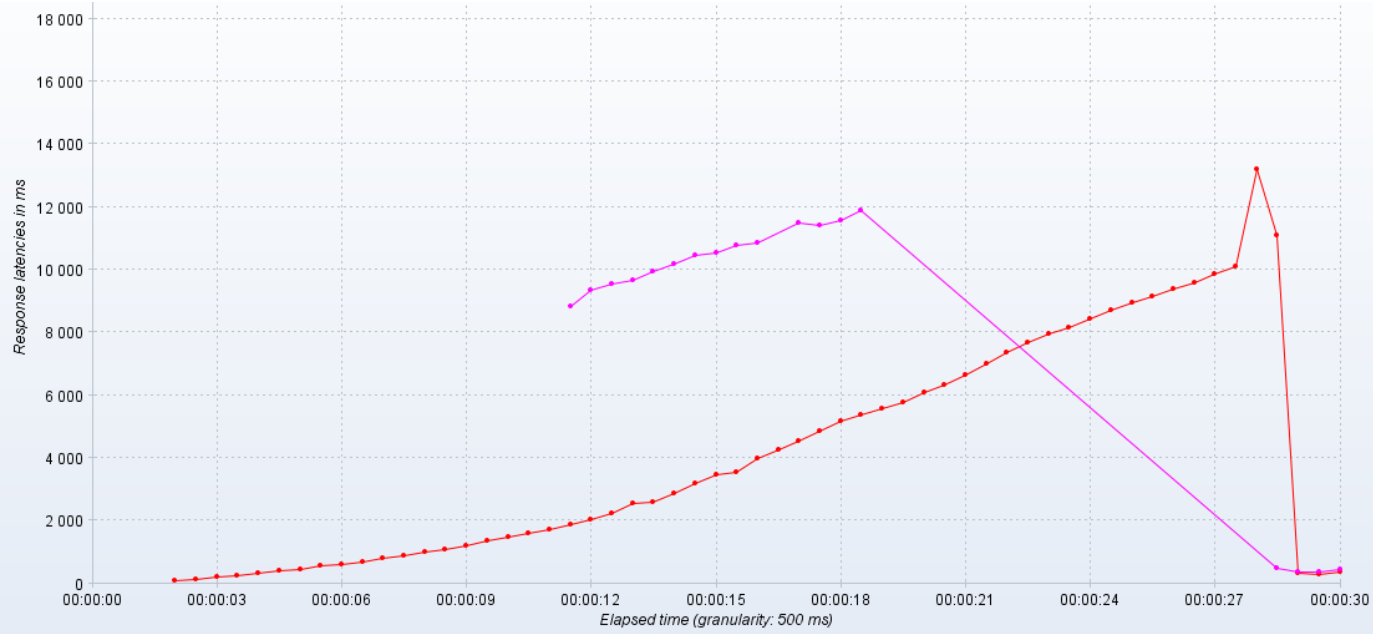
Number of Loops: 3



(3) Number of threads(Users): 800

Ramp-up time: 1 sec

Number of Loops: 3



Latency is the difference between the time when the request was sent and the time when the response has started to be received. HTTP/2 performs better than HTTP/1.1 over high-latency connections. This is because the binary framing and header compression built within the new version of the protocol makes communication more efficient and require fewer round trips but for analysis rambase use HTTPS which include SSL (Secure Sockets Layer) and it is a security protocol that's used for HTTPS. Most studies show that HTTPS is slow because of the computational requirements of the security protocol. Thus HTTPS/2 takes more latency than the HTTP/1.1 whether the number of threads is high or less.

It can be concluded that latency time is one of the overheads for HTTPS/2 further above results shows that HTTPS/2 takes some time to start the response when the number of threads is more, such as HTTP/2 place the first response at 00.00.11 ms while testing 800 threads. (node project: https://github.com/HansikaW/http-2analysis)

Summarized Load Test Result Comparison

	Number of Threads	Evaluation Factor	HTTP/1.1	HTTP/2
Response Times Over Time	20	**Mean Time(in ms)**	341.95	180.31
		Max Time(in ms)	512.36	301.57
		Min Time(in ms)	62.83	49.46
	100	**Mean Time(in ms)**	2400.01	1260.80
		Max Time(in ms)	3271.75	1931
		Min Time(in ms)	278	229
	800	**Mean Time(in ms)**	12549.94	8243.62
		Max Time(in ms)	23915.96	11917
		Min Time(in ms)	557	358
Response Latency Over Time	20	**Mean Time(in ms)**	184.35	173.66
		Max Time(in ms)	331.8	306.4
		Min Time(in ms)	26.83	39.87
	100	**Mean Time(in ms)**	1113.32	1260.74
		Max Time(in ms)	1956.29	1923.24
		Min Time(in ms)	81	229
	800	**Mean Time(in ms)**	4141.06	8242.16
		Max Time(in ms)	13222.07	11917
		Min Time(in ms)	119	358

Stress Test Analyses:

This analysis compared the response time and the latency of HTTP/2 vs HTTP/1.1 with many concurrent users and shows how Rambase performs with HTTP/1.1 protocol (D) and HTTP/2 protocol(d).

For a clear comparison, the below results are considered a maximum of 100 users with 3 loops and ramp-up time of 1 sec.

Following Threads scheduling parameters were set to conduct the stress test

- The thread group starts with 800 threads, then start 20 threads, next add 20 threads in every 10 sec by using 2 sec ramp-up time then hold the maximum number of threads for 20 sec. Finally, stop 30 threads every 10 seconds. Therefore, considered Elapsed time is 0 - 00.01.40 ms

Thread the schedule graph for the above parameter:

Active Threads Over Time: shows how many active threads are there in each thread group during the test run

Response Time Over Time: It can be seen that the HTTP/2 protocol always consists of less response time. *Response time comparison under stress test with 0-800 users can see here.

Response Latency Over Time:

As mentioned earlier, response latency is one of the overheads in HTTP/2, thus this graph represents that in most cases the response latency of HTTP/2 protocol is higher than HTTP/1.1

Transactions Per Second:

This analysis counts for each second the number of finished transactions of HTTp/1.1 and HTTP/2 protocols and results is encountered as stress testing because this shows how Rambase performs the transaction with concurrent users with HTTP/1.1 protocol (D) and HTTP/2 protocol(D).

Above graph display for sampler the number of finished transactions per second. Since both lines of the above graphs show the same behavior with near hit points, the following pie chart provides the most appropriate comparison.

Summarized Load Test Result Comparison

	Evaluation Factor	HTTP/1.1	HTTP/2
Response Times Over Time	**Mean Time(in ms)**	1466.78	816.59
	Max Time(in ms)	3049.67	2325.57
	Min Time(in ms)	83	27
Response Latency Over Time	**Mean Time(in ms)**	814.45	818.84
	Max Time(in ms)	1987.71	2325.57
	Min Time(in ms)	31.4	25.75
Transactions Per Second	**Mean Transaction**	20.92	21.32
	Max Transaction	31	37
	Min Transaction	10	10

Lighthouse to Test Page Load Performance

Lighthouse is an open-source, automated tool for improving the performance, quality, and correctness of web applications. When auditing a page, Lighthouse runs a barrage of tests against the page and then generates a report on how well the page did. Lighthouse auditing conduct page load performance test such as time to interactive, latency, speed index, resources optimization, Time To First Byte, scripts execution time and so on. Chrome extensions negatively affect page load performance; thus performance audits are carried with incognito mode and for more accuracy, the most frequent results of the sample test are encountered here with generated web page performance trace.

Performance Result Comparison

	HTTP/1.1	HTTPS/2

Performance Score	73%	78%
Speed Index	3.5 s	3.1s
Load Time	97 ms	98 ms
Scripting Time	2162 ms	1187 ms
Rendering Time	276 ms	108 ms

The above table shows that HTTPS/2 has performance gain than the HTTP/1.1 further scripting and rendering time taken for HTTPS/2 shows relative shortage because of HTTP/2 ServerPush feature support for a push all the relevant scripts, styles and images without the client having to request each one explicitly.

Page Load Performance Result for HTTP/1.1

Tested URL: http://localhost:1640/default.aspx#/COA

Page Load Performance Result for HTTPS/2

Tested URL: https://localhost:44392/Default.aspx?target=JHEVSYS#/COA

Conclusion

After considering all the analyses, It can be seen that the latency time of HTTPS/2 is high because latency time is overhead of HTTPS. Although the response time of HTTPS/2 shows less tendency than HTTP/1.1. Thus conclusion can be drawn as the HTTP/2 protocol has the best performance.

Ref

Node project with https/2 serverpush feature : https://github.com/HansikaW/http-2analysis

Clientframework with http/2 serverpush feature: https://github.com/HansikaW/clientframework

Cloud computing

- Definition - https://martinfowler.com/bliki/CloudComputing.html
- Cloud design patterns - https://docs.microsoft.com/en-us/azure/architecture/patterns/
- Microservices
  - https://martinfowler.com/articles/microservices.html
  - http://microservices.io/
  - https://samnewman.io/books/building_microservices/
  - Google microservices demo - https://github.com/GoogleCloudPlatform/microservices-demo
  - Microservices by Sam Newman - https://www.youtube.com/watch?v=PFQnNFe27kU
  - Monolith decomposition
    - https://www.youtube.com/watch?v=9I9GdSQ1bb
    - https://www.youtube.com/watch?v=64w1zbpHGTg
  - Communication patterns
    - Async - https://www.youtube.com/watch?v=DzGuDNHsOQ
  - Orchestration
    - https://kubernetes.io/
  - Service meshes - https://microservices.io/patterns/deployment/service-mesh.html
    - https://istio.io/
    - Azure fabric - https://azure.microsoft.com/en-us/services/service-fabric/
    - AWS App Mesh - https://docs.aws.amazon.com/eks/latest/userguide/appmesh-getting-started.html
- Certifications
  - AWS - https://aws.amazon.com/certification/
  - Azure - https://docs.microsoft.com/en-us/learn/certifications/browse/
  - GCP - https://cloud.google.com/certification

Persona sketches

Picture

Demographics

Age : 45

Occup : Team Leader

Gender : Female

Behaviors

Music lover

Cartoon lover

Smooth handling.

Slow learner.

Using Spectacel.

Key Board burn

Goals & pain points

- Check Supply Chain.
- Overlook Production line test.
- Quality Check
-

Picture

Demographics : Empathized User

Production line Technician.

Age: 40

Gender : Male

Computa litteracy: Normal. (Application).

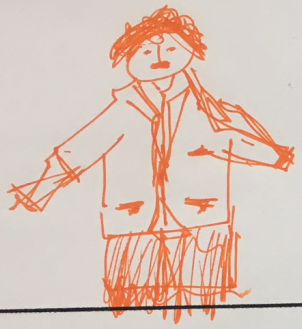
Behaviors

- Workaholic
- Young
- Party
- Fast decision maker
- Short Tempers

Goals & pain points

- Track an order
- Forward an order
-

Picture



Demographics

Age : 40 - 45

Gender : Male

Location : Vatles , Norway

Occupation : Production Manager.

Behaviors

Leader .

Traveler

Reach flexible

Quick in work.

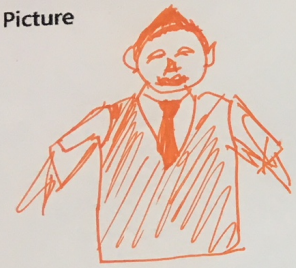
workoholic

Quick Responder

Goals & pain points

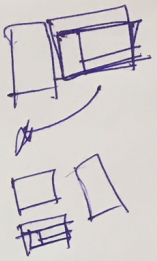
- He Should Able to see all the task overall
- Sending Notes to Production line (Individually / Group)
- Check the Invoice / Quotation / Bill
- Mark Production line priorities
- Able to send item request in ~~prio~~ priorities
- Put orders to outside Products.

Picture



Demographics

: Frank Vigne.
occ: Business Analyst
Age : 45.
Ge : Male.
Relig :
: Married

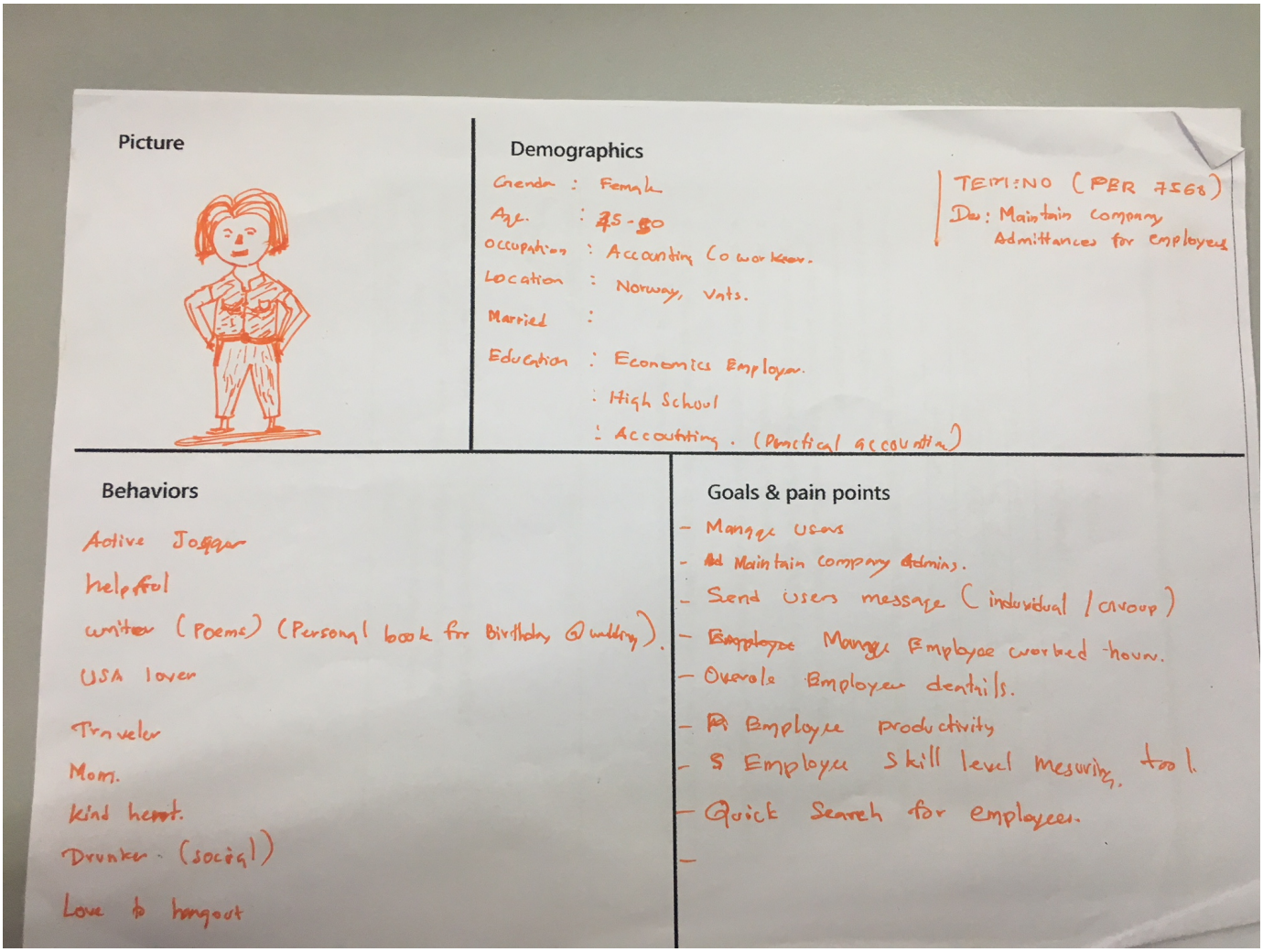


Behaviors

- Traveler
- Family oriented
- Drinker
- Have IT knowledge.
- ~~Cool guy~~ He is not a cool guy.
- Romantic

Goals & pain points

- Create orders to customers.
- Create orders internally
- Create orders
- Follow Orders.
- Inter company function.



Key Points

1. All Features must be keyboard friendly
2. All Features must be usable in Mobile and Desktop

Revamping UX

What is UX

Objectives

KPI

Responsibilities

How do we provide better UX to customer

USP and UVP

CF Team

RPS Team

Synergizing Sessions

Synergizing Session 24/05/2019

Onboarding

General Induction

Rambase Application Development

Rambase App Development FAQ

General Induction

Welcome to the team Hatteland! our project has the following sub-teams as per below,

- Client framework
- App development
- Printing and integrations
- RACR and core migrations

First of all please make sure to complete all action items,

- [x] Connect to the company's network. you can either use WiFI or LAN cable
- [x] Make sure you are logged in to company email with Outlook
- [x] Make sure you have set a 16 character password for 99x Miscrosoft account
- [x] Make sure you have activated two-factor-authentication (2FA) for 99x Microsoft account Multi-Factor Authentication Guidance [Internal Use]
- [x] Sign into seranet which is the company area on Atlassian
- [x] Install required general software/tools such as Git, Microsoft Teams, Visual Studio, Vscode, etc
- [x] Create an account on Bitbucket
- [ ] Activate 2FA for your Bitbucket account https://rambase.atlassian.net/wiki/spaces/RDTL/pages/1049198639/Two-step+verification+in+Bitbucket
- [ ] Say hello to the customer
- [ ] Send full name, company email, Bitbucket username, etc to the customer when they ask
- [ ] Make sure you have access to Hatteland spaces on Atlassian, Teams, and Bitbucket
- [ ] Make sure customer provides you VPN access
- [ ] Enable OTP by asking Ruben

General knowledge about the platform and ERP ecosystem

- Rambase general architecture - https://rambase.atlassian.net/wiki/spaces/RCF/pages/647856565/Client+Framework+Architecture+Documentation
- Official website of Rambase - https://rambase.com/
- Parent company website - https://www.hatteland.com/
- Platform documentation for the users - https://help.rambase.net/
- What is ERP software - https://www.youtube.com/watch?v=Da1hUqzoiAo
- Maturify learning module about Hatteland and Rambase - https://app.maturify.com/app/teams/team-profile/5bfe0feafc4d2200019ab4a1/model-progress/6108dc9f95a7470008ffd16d/model/overview
- YouTube channel of Rambase for tutorials - https://www.youtube.com/channel/UCFSFJVzd4Esc4TwzMSaaf7w
- Also, Please check the Rambase wiki when you get access to it for architecture documentation.

Please complete all the Maturify models in the Hatteland group https://app.maturify.com/app/teams/team-profile/5bfe0feafc4d2200019ab4a1

Security awareness program

GDPR: A quick introduction: GDPR compliance guide

Privacy awareness: https://seranet.atlassian.net/wiki/spaces/IS

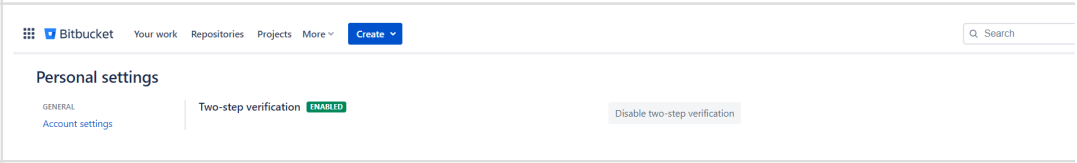
Team specific getting started guidebooks

App development: Rambase Application Development

Good Luck.. happy coding!

Onboarding security checklist

	16 character password for Microsoft account	2FA for Microsoft account	2FA for Hattel and tenant	2FA for Bitbucket
Samudra Kanankearachchi	☐ Done and verified	☐ Done and verified	☐ Done and verified	☐ Done and verified
Hashan Wanniarachchi	☑ Done and verified	☑ Done and verified	☑ Done and verified	☑ Done and verified
Sachith Jayasinghe	☑ Done and verified	☑ Done and verified	☑ Done and verified	☑ Done and verified
Obhasha Priyankara	☐ Done and verified	☐ Done and verified	☐ Done and verified	☐ Done and verified
Warsha Kiringoda	☑ Done and	☑ Done and	☑ Done and	☑ Done and verified

	verified	verified	verified	
				
Savindu Bandara	☐ Done and verified	☐ Done and verified	☐ Done and verified	☐ Done and verified
Sahan Jayasinghe	☐ Done and verified	☐ Done and verified	☐ Done and verified	☐ Done and verified
Dharana Rodrigo	☐ Done and verified	☐ Done and verified	N/A	☐ Done and verified
Fazna Fousen	☐ Done and verified	☐ Done and verified	N/A	☐ Done and verified

Rambase Application Development

Rambase application developer creates HTML5 applications on top of RCF(Rambase Client Framework).

This tutorial guides developers in making HTML5 applications for the Rambase platform.

1.1 Required technical knowledge

- Javascript Basics (JSON, Promises, etc..)
- Typescript
- AngularJs
- HTML/LESS
- Git

1.2 Required tools

- Rabbit - For creating/searching components

https://updaterabbit.rambase.net/ (VPN should be enabled to access)

- Homer

```
$ npm i -g rb-homer
```

- Visual Studio Code (Recommended)

https://code.visualstudio.com/download

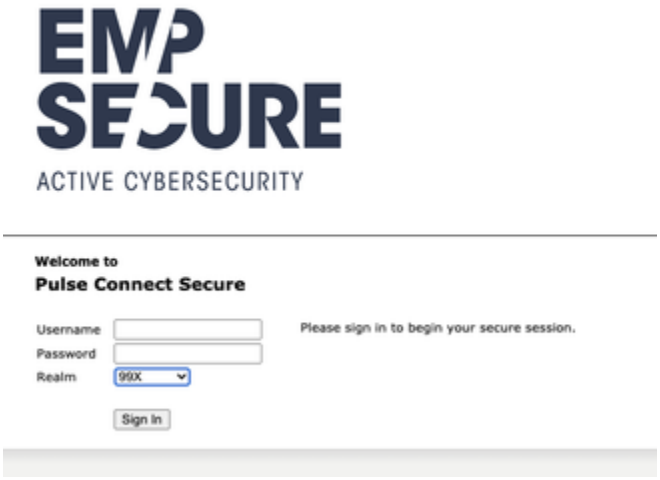
1.3 Developing Rambase applications

Documentation 1 2

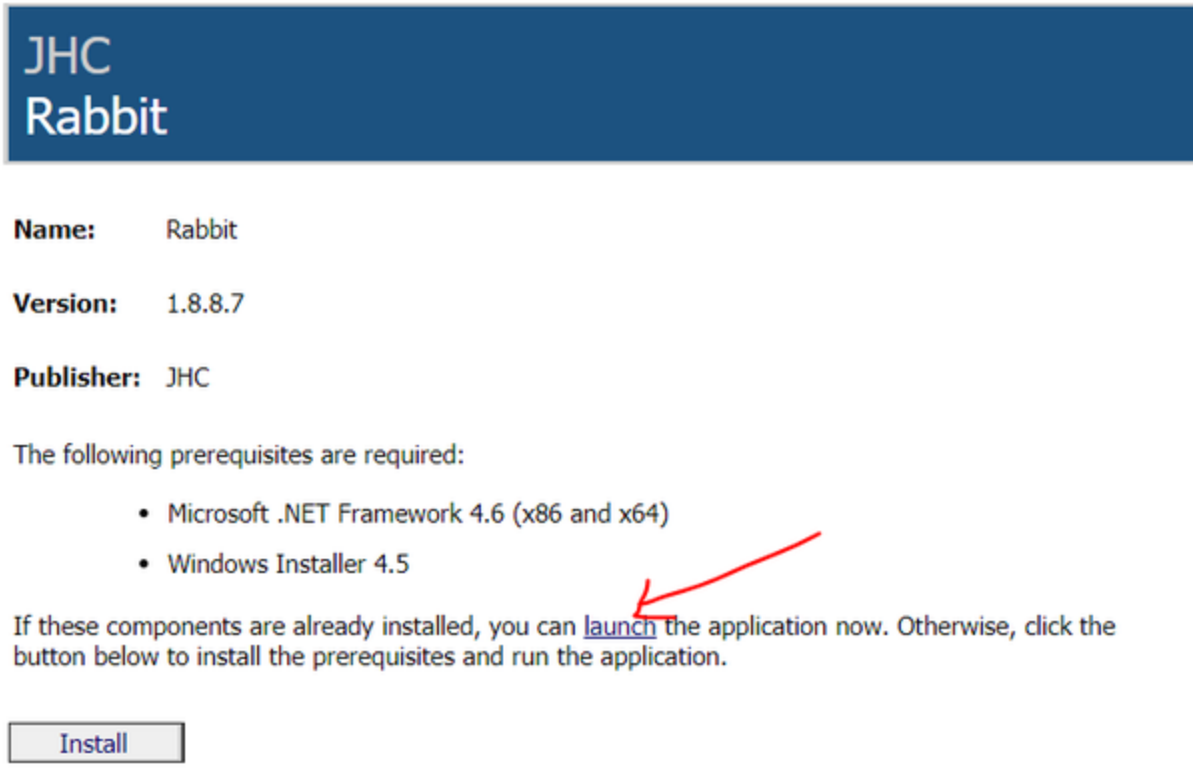
Rambase applications consist of several components which are currently written using AngularJs with Typescript.

1.3.1 Configure VPN

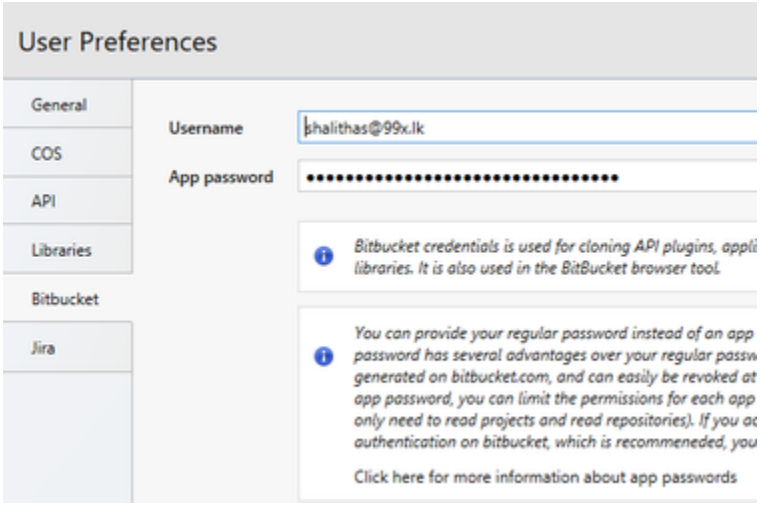
1. Go to https://vpn.empsecure.com/



1. Sign in to start VPN.
2. After VPN is enabled go to https://updaterabbit.rambase.net/
3. Install dependencies(will be listed) if not installed.
4. Click on Launch.

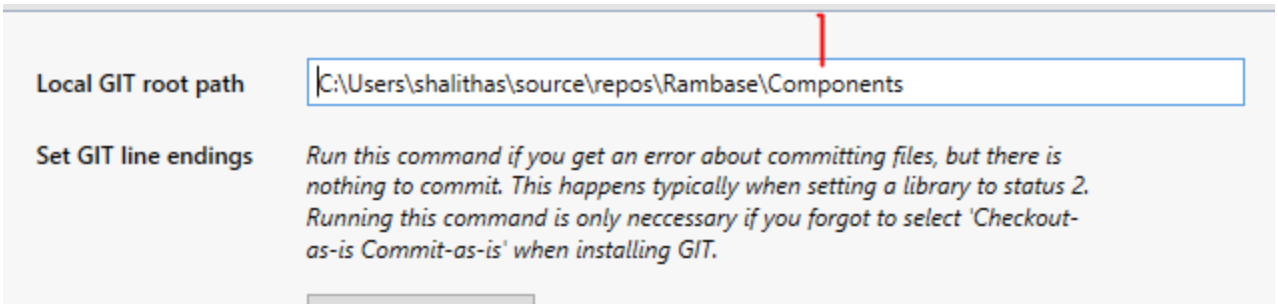


1.3.2 Configure Rabbit



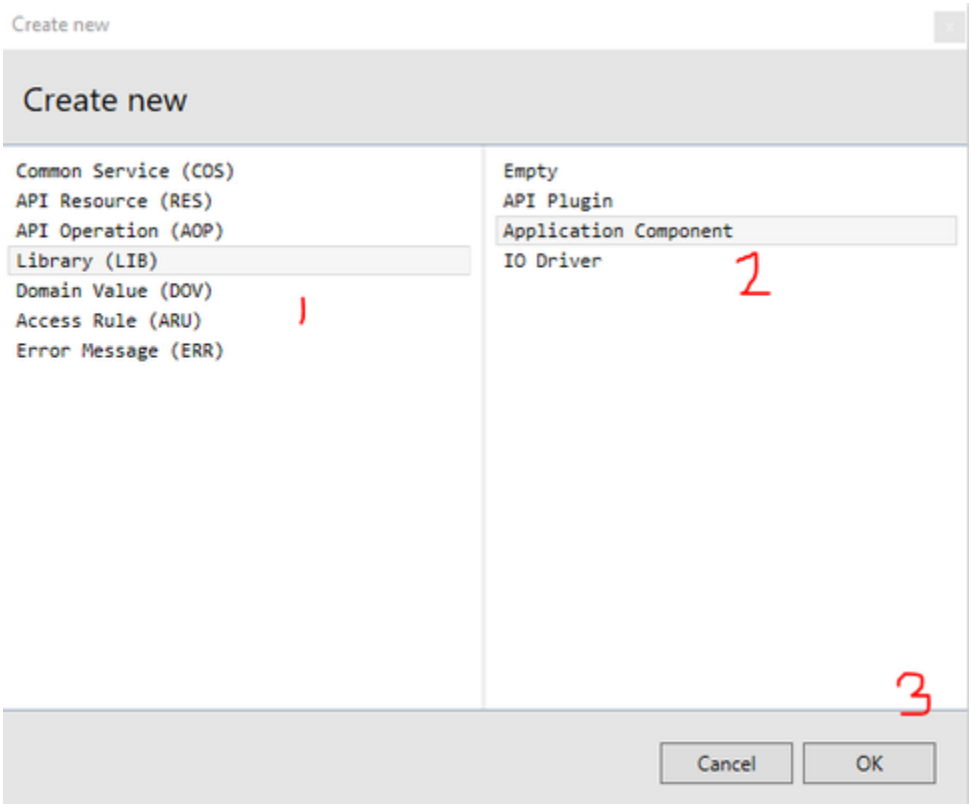
Before you begin anything make sure to add your Bitbucket credentials to the Rabbit.

App password doc (https://confluence.atlassian.com/bitbucket/app-passwords-828781300.html)

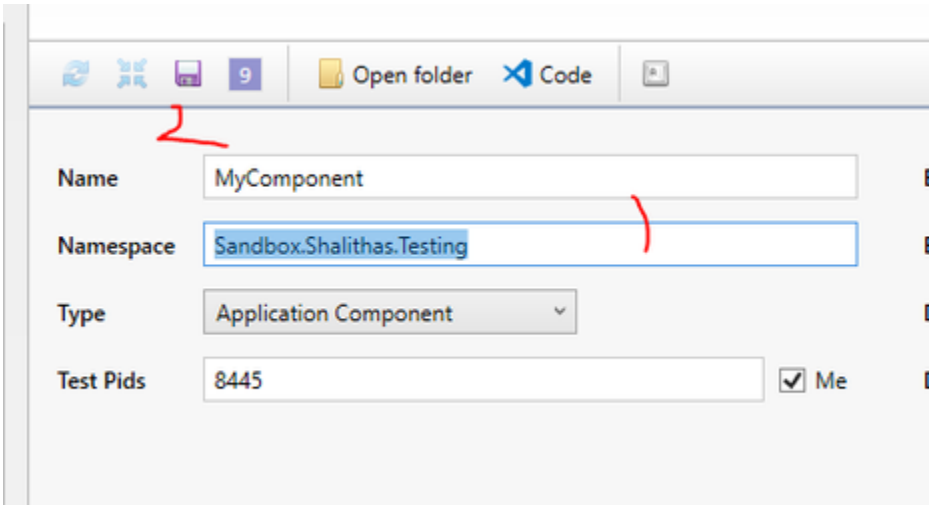


Also select a directory where you wish to save your components locally.

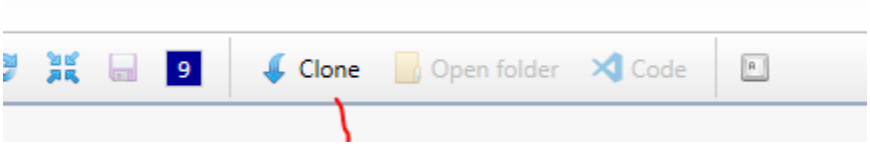
1.3.3 Creating a new component



Go to **File -> New** and Select **LIB** resource and then select **Application Component.** Thereafter click **OK** button to continue.



Enter a name for your new component and fill the namespace (Note: If you are doing some experiments it is recommended to use **Sandbox.YourName.Testing**). Click on the **save button** or press **Ctrl+S** to continue. Thereafter a new Bitbucket repository will be created.

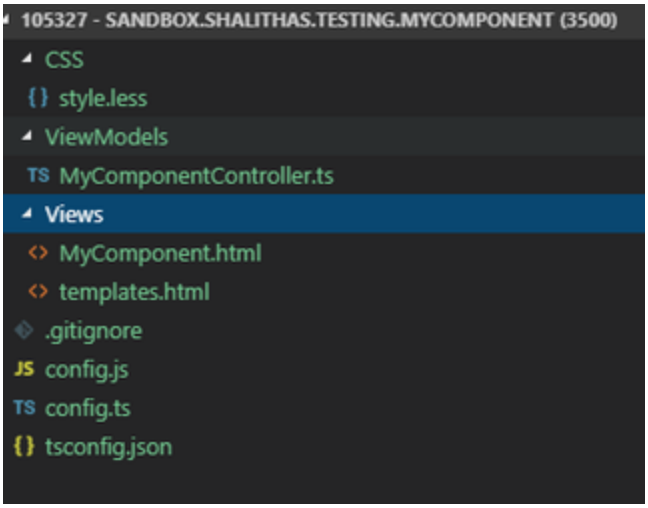


Click on the Clone button to set up a local Git repository for your new component. Thereafter Rabbit will clone the component source into **<yourComponentsDir>/Applications**

Open the component source with Visual studio code. Initially, it is an empty repository. Therefore you can generate the basic component template using **rb-homer**

```
$rb app

// Type‘yes’ and hit enter

$ git add .

$ git commit -m ‘Initial version’

$ git push origin master
```

1.3.4 Component Structure



File	Usage
Style.less	SAAS stylings for your component
MyComponentController.ts	AngularJs Controller
MyComponent.html	AngularJs View
templates.html	View area to hold your Kendo windows etc.
config.ts	Component configuration **dataSources**: Resource endpoints of the REST API **Parameters**: Communication mechanism between components.

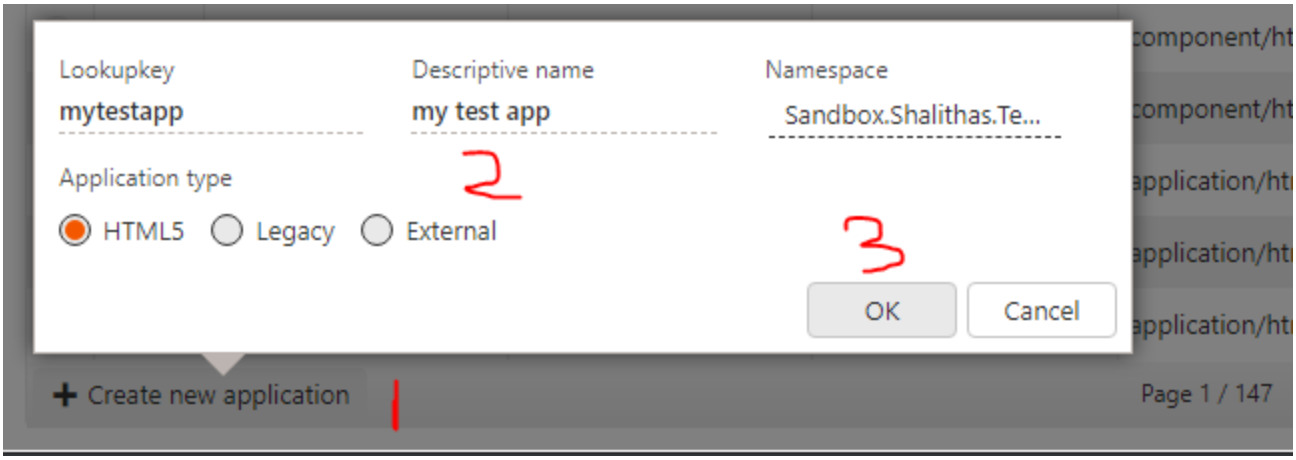
Please notice that you can create classes and interfaces accordingly matching your component's complexity. We are still in the process of creating a development guideline for components development.

1.3.5 Testing your component

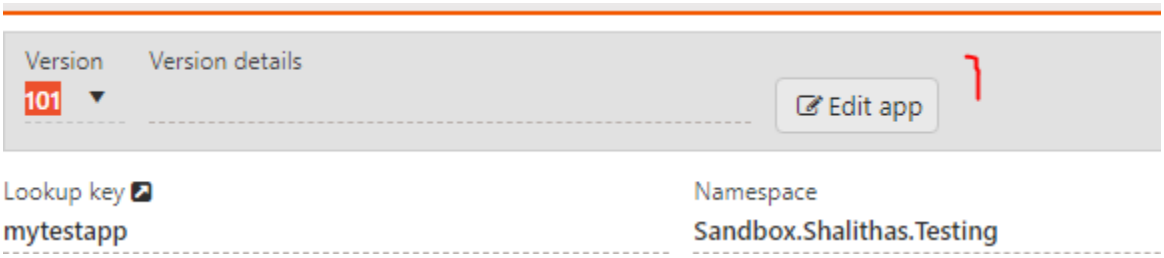
If you need to test your component you need to create a Rambase application.

Login https://www.rambase.net/

Type **app** and hit enter. Thereafter, App Editor application will be loaded.



Click **Create new application**, fill the details and click on the **OK** button to create a new app for you.



Click **Edit app** from the right side to start editing your application. Before adding the component your component needs to be zipped and also needs to be stored in the file server. So, let's use **rb-homer**

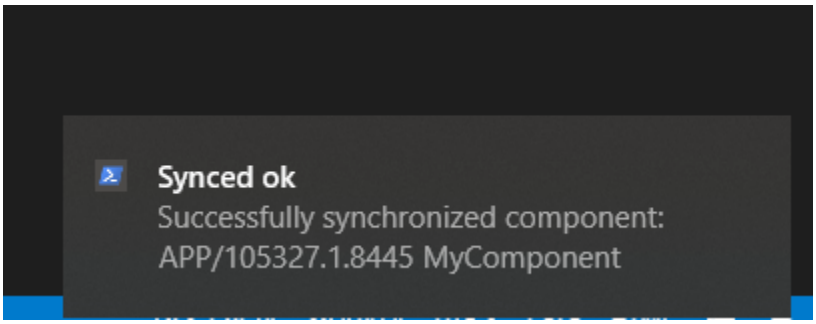
Go to the **<yourComponentDir>/Applications** or into your cloned component via terminal and enter,

```
$ rb listen
```

This will sync your local component changes with the file server.

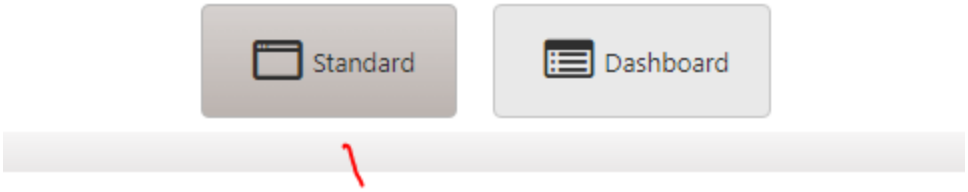
Note: If ***$ rb serve*** is used application page will be reloaded automatically to see the latest changes. (Use ***?debug=true*** query parameter and click ***play*** button at top right to enabled live reload feature)

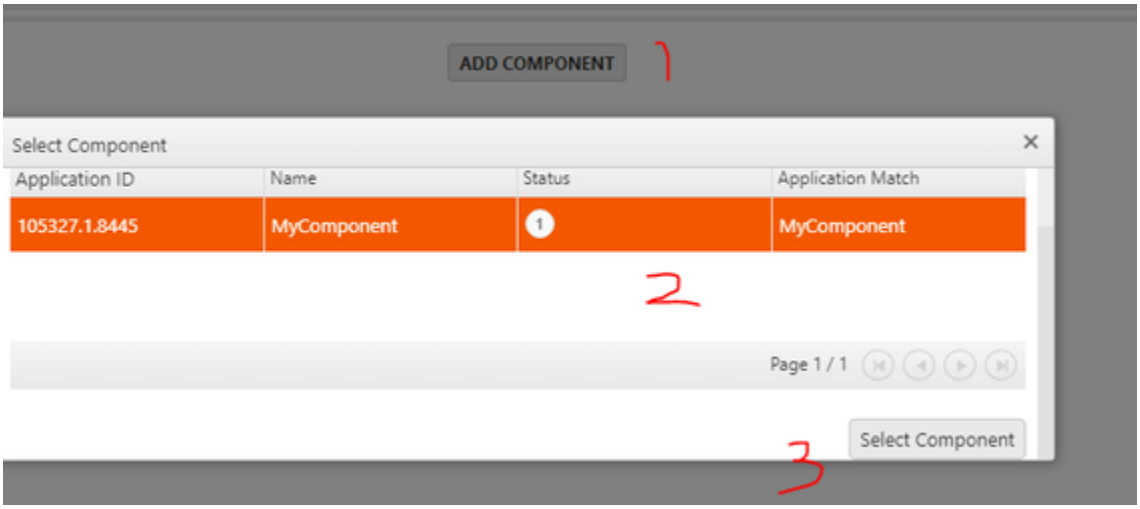
Press Ctrl+S from Visual studio code.



If you got this message from the right side of your screen. Everything looks fine.

Got the App Editor application again and start editing your test application.

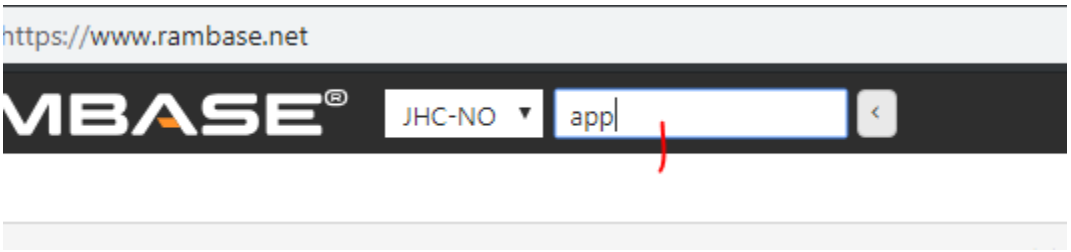




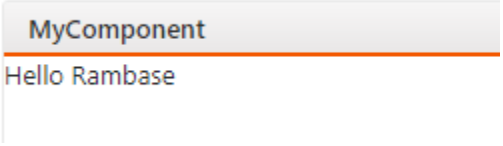
Click on the **Standard** button since we are going to create a normal Rambase application.

Click on the **Add Component** button, search and find your new component and thereafter click on the **Select Component** button to continue.

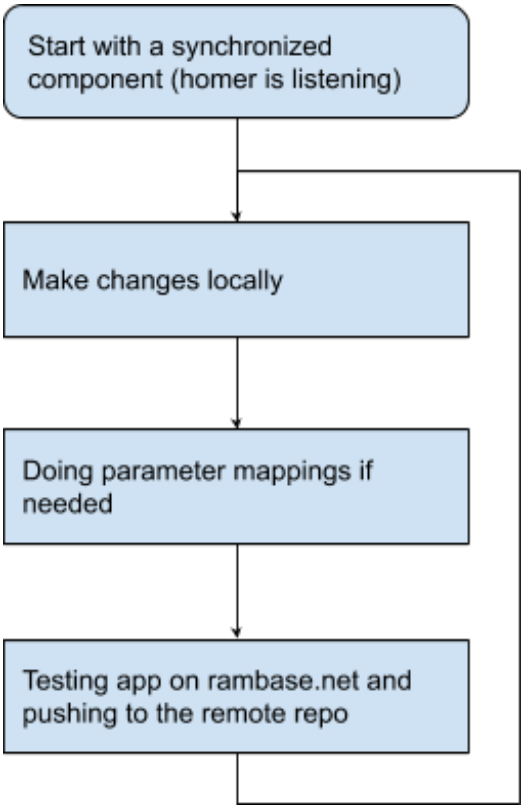
Hit Ctrl+S to save your application



Select **TEM-NO** database and navigate in to **MYTESTAPP** (case insensitive) to see your component.



1.3.6 Component Development workflow



Notice. This sample application is only visible to you since the application and its component still in status 1. If you want to allow some other users to access your application you need to set the application and its components to status 2. Also, if you wish to enable your application for all users in the current system status 4 needs to be used.

Read more about versions here

Common standards to follow

Rambase App Development FAQ

As a Rambase application developer, what are my working fields?

- Frontend development: Creating applications and components with HTML, CSS (less), TypeScript, and Angularjs.
- Backend development: Creating API endpoints with XML, implementing APIs with COS, developing API plugins with C#, and developing batch jobs with COS.
- Mobile development: Developing the official Rambase mobile app (hybrid) with Xamarin and C#.

What I need to learn for creating Rambase applications?

- The app editor application (Go to the APP app and try to open an app with the app editor)
- TypeScript: (Basic types, classes, exports and imports)
- JavaScript: (Web APIs, language features like arrow functions, variable definitions, iterations, conditions, etc)
- Angularjs: Check the MVVM pattern, controllers, two-way binding, directives, template syntax
- Rambase framework: widgets, services, component lifecycle APIs, etc
- Kendo UI widgets: We use Kendo UIs directly if the required widget is not wrapped in the Rambase API for unique requirements.

What I need to learn for developing APIs and backend components

- RES: Learn Rambase resources XML syntax
- COS: Learn COS by developing some test programs in the sandbox and looking at production COSes
- AOP: Actions that doesn't belong to and CRUD endpoint i.e, Activating a customer https://rambase.atlassian.net/wiki/spaces/RAB/pages/152898657/API+operations+AOP (Also refer to this document)
- ARU: Simple condition-based permission checks for POST, PUT endpoints, context menus, AOPs, tabs, etc

- API plugins: C# based code for endpoints and AOPs
- Batch jobs: Long-running jobs, output jobs (printing and sending email, etc), and cron jobs (daily running batches etc). Each batch job has two elements: FRM and IOD. FRM is the configuration and IOD has all steps (COSes)

What I need to learn for developing Rambase mobile app?

- Xamarin (not Xamarin Forms - the app use platform SDKs directly with C#) and C#
- Webview APIs in Android and iOS
- Azure Push notifications
- Android and IOS platform SDK APIs
- RESTful API authentication principles
- Hybrid app concepts (Check how Ionic works)

I get some toaster errors about permissions - how to update my app permissions?

- Go to the ROLE app on TEM-NO and add the required roles, then activate permission changes via CMI

In which apps, the app team usually work?

- CUS
- SUP
- CNT
- TSK
- OPP
- TSP
- WHL
- WLA

What is the easiest way to debug an API?

- Add an exit() call or throw() call with a message from the COS and call the API from Rabbit's RESTester

How does a RESTful API call works & what's the flow?

- Application client Client library APIs HTTPs request API gateways and networking related components API server instance RES definition COS COS runner SQL Server instance

Is there any in-built way to create CRUD related COSes automatically?

- Yes, In rabbit you can go to file->New-> Common service(COS) and select the required COS type. This will automatically create a skeleton for your COS that you can modify anyway you wish to.

What if I want to delete my edit version of a LIB or a COS?

- Click the red 'X' icon that will delete your version, then click the '<-' revert icon to discard any uncommitted local changes. This 2nd step will remove your PID from the commit list.

Are there any custom grid-like component made with ng-repeat?

- Yes, we call them ng-repeaters. Please check the WHLWorklogList component

How to send a value from a component to another component?

- Try to use URL parameters via the app editor. If you need to send events, use Angularjs broadcasting. For rare cases we use sessionStorage and Angularjs app scope too.

Can I use the same component twice in the same app?

- No, the client framework doesn't support duplicated component parameters, but you can apply a workaround by dropping params to the Angularjs app scope. Check WHL's edit period window implementation

Can I call another component inside a popover?

- No, as of now, this isn't possible. Popover has to be one component. Only way to create a clear division between UI component like areas that goes inside a complex popover is to create separate html for each section.

How to find component names in a particular app?

- We made a Chrome extension for it. If you click on the extension icon, it will display all component names.

How to create a new API? what's the order?

- First of all, make sure all archive fields are in status 3 or 4. If not, contact the dictionary team and ask.
- Create resources first starting from GET (Eg: sales/customers/{customerId}) and GETLIST (Eg: sales/customers)
- Add ARUs for POST, PUT and DELETE endpoints
- Make sure to add field descriptions, DOVs (domain values), default values, minimum values, etc.
- Create COSes for CRUD operations via Rabbit COS templates (File New)
- Add validations and add API COSes
- Link API COSes to resources
- Use the analyzer tool and fix all issues with resources (Click on the "Analyze button")
- Test the new API by creating some records
- Finalize by adding object mappings and assigning document tree nodes

What are objectType mapping & Documentation tree mapping?

An archive has an object type (ex: archive=TSK objectType=Task). Each table of an archive also has an object type (ex: Table=T1 objectType=TaskTag). Each table-objectType should have a mapping with the resources. This mapping is objectType mapping. To check various object type mappings go to 'help' in rabbit and click 'Archive<->ObjectType Mappings'. If a new table is created, a new ObjectType mapping must be added.

In RES go to documentation tab. To be able to put a RES to ST2 you must have a documentation tree mapping. If this RES is in a newly created path, a new documentation tree mapping must be added.

How can I create new ObjectType mappings and Documentation tree mappings?

You need to contact the responsible person in the resource team and ask them to add it.

How to create a shared project?

When you're doing changes in multiple RES/ COS/ LIBs, to make them easy to deploy, create a shared project by going to projects tab +Add. You can also add shared projects created by other's in to your tab.

How to enable custom fields?

RES - fill the 'custom field archive' field. ex:

COS - add UDF to arg/ res

Component - create a generic component for custom fields.

How do identify the current active named filter from the COS level?

The special design pattern used for this is called **legacyFilter.** This can be defined at the resource level and can be used from the COS level

Example usage

- In the resource level under the filter section, we can define

```
legacyFilter('CONTAININGEMPLOYEEIAPPROVE=="1"')
```

- To identify the filter, you should use a common API and there is a shared method called **GetBooleanReplacedFilter.**

```
( ARG.FILTER ) = API::GetBooleanReplacedFilter( ARG.FILTER,
"CONTAININGEMPLOYEESIAPPROVE":FIELD, 1:TRUEVALUE );

 if( ARG.FILTER .=. "CONTAININGEMPLOYEESIAPPROVE" )
 (
  // To Do if the mentioned filter is active

  ( ARG.FILTER ) = API::ReplaceFilterFieldByOperator( ARG.FILTER,
"ContainingEmployeesIApprove":FIELD, "==":OPERATOR, F:EXPRESSION );
 )
```

- **ReplaceFilterFieldByOperator** can be used to replace the filter with the operator and give the expression

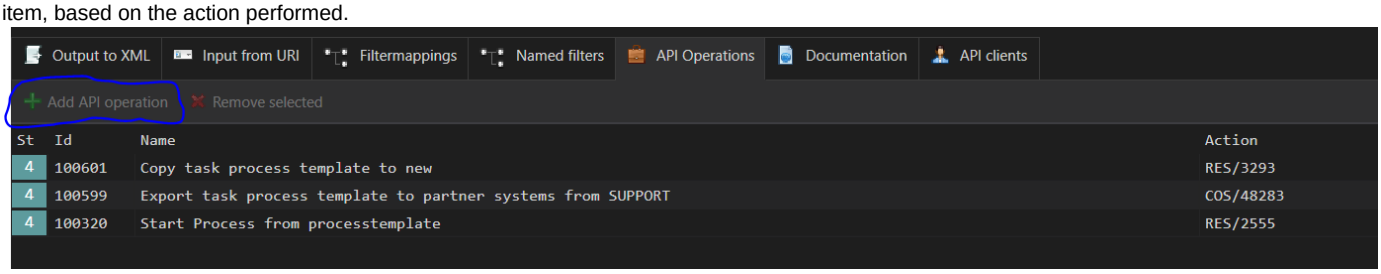
Can we only get/ iterate a given number of records like 80 or 300, when getting data from an archive

Answer: No. We can get all the records as well.

https://rambase.atlassian.net/wiki/spaces/~612718332/pages/3045851137/Iterate+all+over+the+data

Why my created AOP's auto generated URI is showing as "System/api/…"?

Refer to the documentation https://rambase.atlassian.net/wiki/spaces/RAB/pages/152898657/API+operations+AOP. After creating the AOP you have to also add the AOP id to the correct resource's "Api Operations" tab (This part is not in the Rambase documentation). Only then the AOP's URI will be correctly updated. Else the generic URI will be there ("System/api/…"). Resource should be always a GET resource. GET all or GET item, based on the action performed.



Adding rb-inputs with ng-switch does not work

At the moment (16/08/2022) ng-switch is not supported by framework to work with the rb-input elements. Instead you have to use ng-if.

Internal Articles of the week - IAOW

IAOW will motivate every member of the team to write blog posts every fortnight. Goal is to publish our own articles with the company blog newsletter - Articles of the week

Add your blog posts here

Blogs posts

Link/Tittle	Category	Blogger
Roadmap To Become An Awesome Open Source Contributor In 2019 https://www.99xtechnology.com/blog/techinsight/roadmap-to-become-an-awesome-open-source-contributor-in-2019/	Open Source	@ Shalitha Suranga (Unlicensed

Internal TechTalk

Internal TechTalk is a weekly tech meeting which will happen every Friday.

What we do at Internal TechTalk?

One team member will do an in-depth tech session (probably a hands on session) focusing on technologies/concepts which are related with project.

Example topics: custom elements, angularjs directives, type script practices

Sessions Log

Tutor of the day	Topic	Resource/ presentation links
@ Hashan Wanniarachchi	Micro-front end architecture and custom elements	https://github.com/HashanMWanniarachchi/NGWebcomponents
@ Hansika Wanniarachchi (Unlicensed)	HTTP1.1 vs HTTP2.0 performance research findings	
@ Athif Shaffy (Unlicensed)	Push Notifications with Firebase	
@ Shalitha Suranga (Unlicensed)	Compiler/Transpiler design basics and introduction to RACR	https://docs.google.com/presentation/d/1z2FCj8hxPD-qXyzY0pzCGpNpcc09Tbcvg0PO62g1MRs/edit?usp=sharing
@ Samudra Kanankearachchi		
@ Rajika Abeyrathne (Unlicensed)		
@ Sachith Jayasinghe		
@ Raashid Ahamed		

Teck talk calendar

Title	presenter	Scope / Description	Date
Rambase Architecture	@ Samudra Kanankearachchi		24/05/2022
VS code extensions	@ Dharana Rodrigo		27/05/2022
	@ Sahan Jayasinghe		01/06/2022
12 factor apps	@ Samudra Kanankearachchi	clean code patterns	

Product Health Review (PHR)

PHR-related pages

Performance Tasks

QA Strategy

Review Notes - Hatteland

Performance Tasks

Options to add performance related tasks on Jira

Performance Tasks Specification

Options to add performance related tasks on Jira

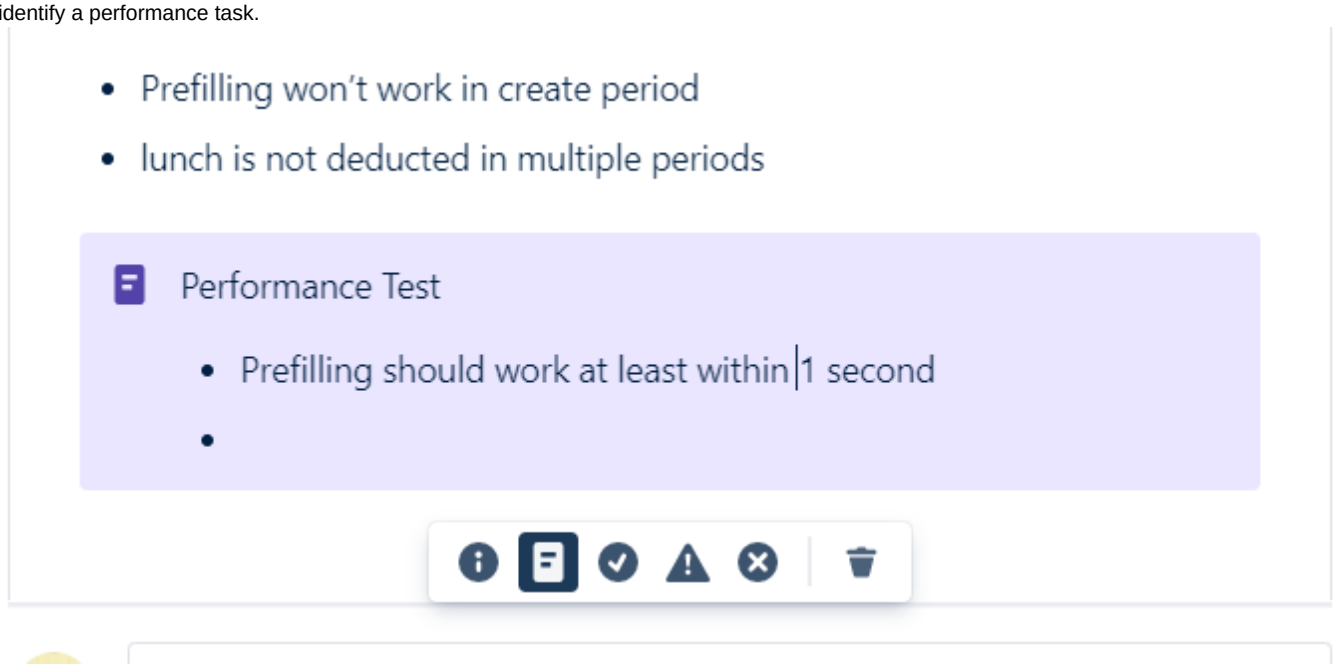
Option 1: Using Jira Test cases

Jira Test cases feature offers a very detailed way to manage manual test cases. Therefore, this feature also can be used to manage performance related tests too.

Pros	Cos
Very detailed report about what should be checked Ability to create Steps per each performance factors	This seems to be a time consuming approach

Option 2: Using description section with different formatting

This is a very simple approach by utilizing task description to add information about performance test criteria. In addition, a label can be added to identify a performance task.



Pros	Cons
This is a very fast method because we often use description section	Assignee has to manually mention what the status of factors

Option 3: Using a checklist plugin

Jira has a marketplace that contains many useful plugins. There are several plugs that allow us to create checklists on tasks. These checklists also can be used to manage performance test criteria.

Pros	Cons
Takes less time than the test cases approach Having similar features as the test cases approach	Adds a bit more complexity

Some popular checklist plugins

https://marketplace.atlassian.com/apps/1220209/issue-checklist-for-jira-free?hosting=cloud&tab=overview

https://marketplace.atlassian.com/apps/1211562/checklist-for-jira?hosting=cloud&tab=overview

Option 4: Task type(using epics) as performance tests

Add tasks with epics for performance tests.

Pros	Cons
Easier to group all performance tests. Easy to identify	Perhaps, it will add a bit more complexity

Conclusion

Option 2 looks easy to implement in all teams because it doesn't require any action which needs to be done by the Jira admin.

Performance tasks specification

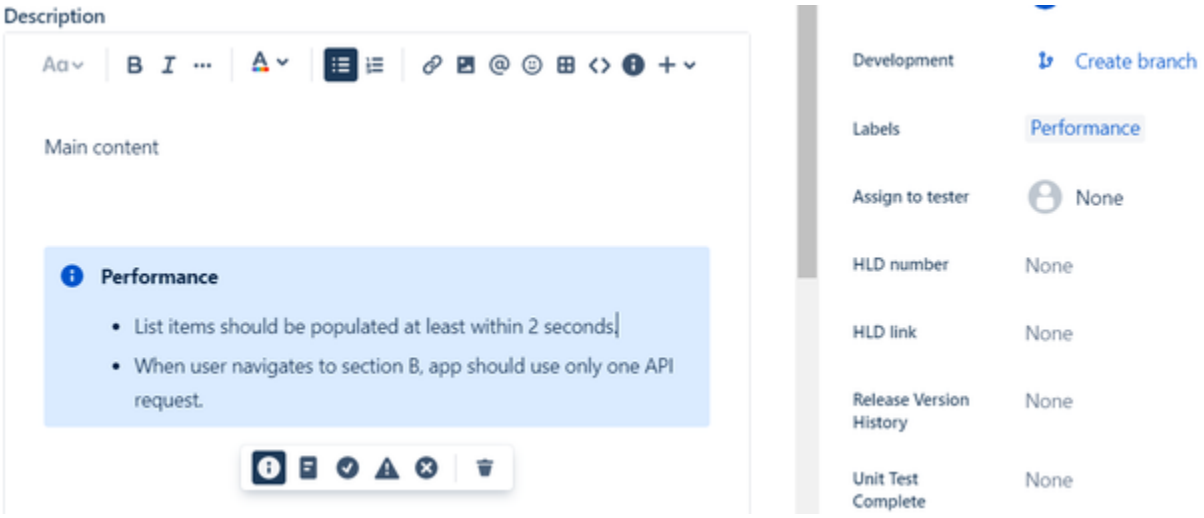
Introduction

If a specific task requires to store performance test criteria, the required requirement factors need to be included into the description of the task with "Info Panel" formatting. In addition, the performance task should use the "Performance" label to identify or group easily. Developers may change "Info Panel" formatting according to the severity of the performance cases, For example. "Warning" style can be used to indicate high severity scenarios and "Info" style can be used to indicate low or medium severity scenarios.

Content

Content of the "Info Panel" should display each performance case by using bullet items. Also, developers can highlight performance related content using "Performance" as the title before the bullet items list.

Example



//Add workload along with performance criteria (ex: in test, stage, prod)

QA Strategy

1. Introduction

This Quality Assurance Strategy presents the main guidelines and the procedures, which are used to ensure the Rambase product quality. And it is targeting the Quality Assurance activities only in Rambase products and their related dependencies. This strategy also describes the overall approach to testing. For each feature or feature combination, will ensure that these features are adequately tested.

When creating this strategy, we consider only Agile Scrum methodology.

*****This document is not a static one, it will be updated from time to time according to the future requirements and application needs.**

The team has 4 main sub-teams working on different parts of the ERP system.

1. Framework team - Focusing on the core framework of the ERP system
2. App team - Develop apps on the system and mobile apps
3. Print team - Delivery/ Output related tasks (Output team)
4. RACR - Focus on the new runtime environment of the system and developer tools like the new CLI tool and VSCode extensions / plugins

Testing approaches and the scopes are different for each sub-team.

Here in this document, we have highlighted the common testing procedure for most of the subteams.

2. Test Scope

In scope

- In Rambase we test every functionality apart from some non-functional testing

Out of scope

- For the initial releases, we are not conducting test automation but in the near future, we will be focusing on that.
- Non-functional testing (performance and security)

3. Test approach

A story grooming session (Work backlog grooming) will be scheduled with the Product Owner when a new story to be implemented. The team will be participating in this session. (since we don't have a QA, developers are keeping a buffer to plan test for each feature)

At the sprint planning, test activities for each story will be planned by the developer. Selected user stories will be tested against the acceptance criteria under this testing type.

A testing task will be created for each story where testing is possible.

Feature release testing:

- Once we have a complete feature, developers and the product owner will be conducting smoke testing on a test state (2) (in Rabbit). In special cases, PO will do code reviews too.
- We are not going to write detailed test cases but a high-level test case document will be maintained. (Probably an excel document)
- Regression suite will be a subset of functional testing which will be **high priority test cases.**
- We are not currently doing performance and security testing.

A Periodic system and integration test will be conducted.

All release tests will be conducted by the product owner.

How to handle and maintain a bug : check the bug verification part of this document

Testing Tools: Visual Studio, SonarQube, Azure DevOps

4. Testing Types

4.1 Unit testing

Unit testing should be written to test each module/feature. All the team members are responsible for doing unit testing. The team uses XUnit as the test framework, Moq for mocking, and VS 2019 as a development tool. Each pull request-wise unit tests run on Azure DevOps build (CI environment).

4.2 Smoke Testing

Each subteam tests whether the happy path is working or not after a new feature addition or a bug fix. This is typically executed just after the deployment, (No test cases needed). If the smoke test failed then the release shouldn't go forward. If successful, continue the functional testing.

4.3 User Story/Functional Testing

Selected user stories will be tested against the acceptance criteria under this testing type.

These should be done under functional testing, if relevant

1.Cross-browser testing - Testing will be mainly carried on the following browsers;

Edge Chromium (latest)

Chrome (latest)

Complete end-to-end testing will not be performed on these browsers separately. The browsers will be selected on an Adhoc basis. Primary Flows will be tested using all browsers when we doing the Feature Testing.

2. Localization - Testing will be mainly done using the English language/culture. If there are any new features added, then it will be tested for translations with the following languages/cultures:

1. Norsk (Norwegian)
2. English
3. Other relevant languages

3. Different OS testing

4. Mobile testing

4.4 Release testing - Product Owner is responsible for doing this

Depending on the impacted areas, if a regression test is necessary, the tester will plan for a Regression test during the release. Regression tests should be started after the code freeze and should be completed before the deployment on the Stage environment to make sure the old features are not broken. This will be done using the Regression test cases written Visual Studio or Azure Devops. Regression time will be captured under a separate testing task for each Sprint.

A smoke test would be carried out for major and minor releases in the stage environment before the production release

Once the Production deployment is done the tester(s) will perform the Smoke test on the Stage and Production environment.

The Testing team will send out a mail to the relevant stakeholders upon completion of the testing in Production.

Root cause analysis will be done on the bugs which are reported by end-users. Tester will analyze the root cause & update the required test case documentation.

4.5 Regression Test

Sub teams identify the functionalities around the new feature (for example: get random a user and test the functionalities around it) in every release. Each sub-team has different regression suites.

4.6 System Test

As a system test, teams test everything in a high-level manner, especially in a major release.

Tests should be written to test the functionality when modules are **integrated**. XUnit can be used to test the integration of different modules in C#.

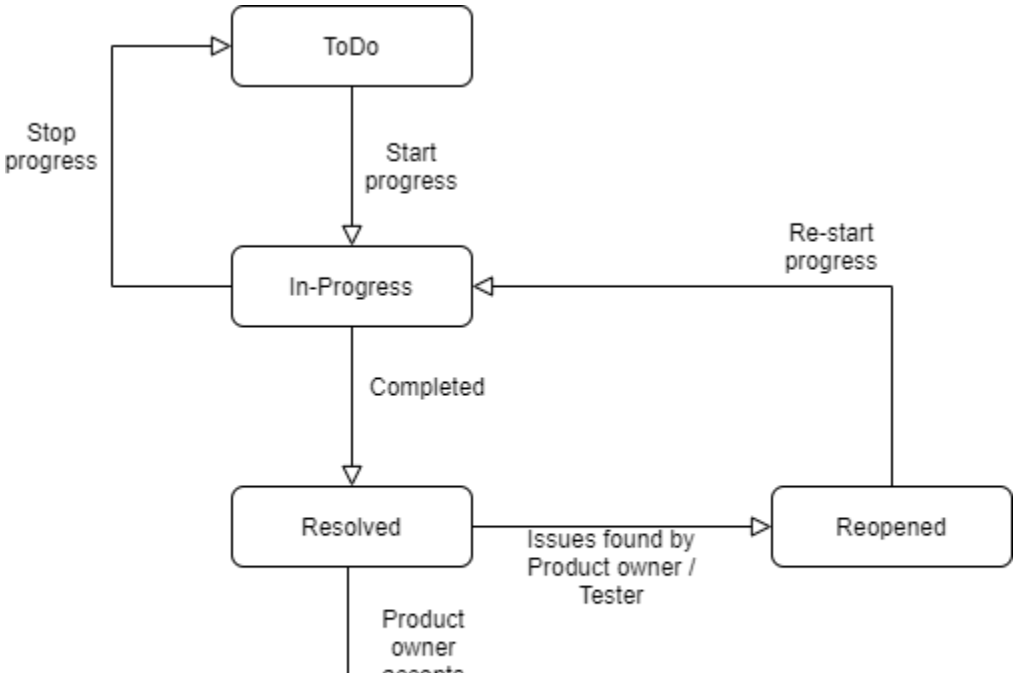
4.7 Product Owner Acceptance testing (UAT)

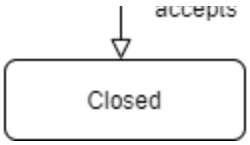
The purpose of POAT is to verify each and every user story which is implemented in the application. After the feature testing is done, QA resources will pass the user story for POAT. Product owner will be responsible for POAT. Using customer requirements, PO needs to test the application function.

5. Bug Verification

Testers will verify the bugs which are in Resolved Status. If the bug is reproducible again(after fixing), the status should be changed to Re-open status. If the bug is not reproducible, it should be changed to Closed status.

The time spent for each bug verification will be captured in the bug itself. A suitable comment should be added to the bug with the relevant details.





6. Test Environments

We will only conduct storage location-related tests for a new feature just to make sure that it syncs with the correct DB. When doing regression tests, we would be choosing the storage locations in an Ad hoc manner and would conduct the tests.

7. Risk Management

In this section, we are describing the risk management aspect of testing the product and how we are going to resolve the issues.

1. No proper test environment for RACR project - customers are working on it.
2. No QA environment (currently they test on dev environment)

8. Handling Special Requirements

Team members should go through these documents and should have a proper understanding of these laws, regulations, and compliances.

- https://rambase.atlassian.net/wiki/spaces/PROC/pages/1334050875/Development+subcontractor+audit+checklist
- https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment
- https://rambase.atlassian.net/wiki/spaces/RBOP/pages/731938877/Group+Policies+in+Rambase
- https://rambase.atlassian.net/wiki/spaces/PROC/pages/1605828611/Release+Management
- https://rambase.atlassian.net/wiki/spaces/RBOP/pages/839450641/External+VPN+Users
- Contract
- https://rambase.atlassian.net/wiki/spaces/PROC/pages/297861606/Guidelines+for+Customer+data+handling

9. Responsibility Chart - QA activities

QA Activity	Responsible Party	Responsibility owner
Unit testing	Local team	Framework team - @ Hashan Wanniarachchi Output team - @ Sachith Jayasinghe App team - @ Warsha Kiringoda RACR team - @ Kalpani Ranasinghe (Unlicensed)
Smoke testing	Local team	Framework team - @ Hashan Wanniarachchi Output team - @ Sachith Jayasinghe App team - @ Warsha Kiringoda RACR team - @ Kalpani Ranasinghe (Unlicensed)
User story/functional testing	Local team	Framework team - @ Hashan Wanniarachchi Output team - @ Sachith Jayasinghe App team - @ Warsha Kiringoda RACR team - @ Kalpani Ranasinghe (Unlicensed)
Release testing	Product owner	Framework team - Ruben Haakonseth Olsen Output team - Ragnar Wiik Johansen

		App team - Kamil Polikiewicz RACR team - Jens Hittenkofer
Regression test	Local team	Framework team - @ Hashan Wanniarachchi Output team - @ Sachith Jayasinghe App team - @ Warsha Kiringoda RACR team - @ Kalpani Ranasinghe (Unlicensed)
System test	Local team	Framework team - @ Hashan Wanniarachchi Output team - @ Sachith Jayasinghe App team - @ Warsha Kiringoda RACR team - @ Kalpani Ranasinghe (Unlicensed)
Product Owner Acceptance testing (UAT)	Product owner	Framework team - Ruben Haakonseth Olsen Output team - Ragnar Wiik Johansen App team - Kamil Polikiewicz RACR team - Jens Hittenkofer

Testing Guidelines

Regression Testing - Framework

Regression Testing - Apps Team

Regression Testing - RACR

Regression Testing - Output Team

Regression Testing - Framework

Framework team does a manual regression testing after each feature implementation or a code improvement or a refactor takes place. For full blown feature implementations, the developer, reviewer and the tester does the regression testing, for code improvements and refactoring, developer and reviewer takes the responsibility.

Steps for the testing are as follows:

1. The developer makes a change in the code by implementing a feature, fixing a bug, improving or refactoring the code.
2. The developer reviews the code and does the regression test based on the area/components that he/she worked on.
    a. This process usually consists of logging in to the application, go to a widely used app and test out the change as part of a workflow (order, crm, user management, etc)
    b. If the change is not part of any identified workflow, it's tested along with some mandatory sub checks*.
3. If the initial testing passes, code is committed and pushed to the relevant branch and the task is put in the review lane.
4. If the code change is visible via the frontend/ or does a significant change to an existing workflow, a tester will do another regression test.
5. Else, review tasks are tested based on its context as a part of sprint completing activities, by a reviewer.
6. Upon successfully completing the test, either the tester or the reviewer closes the issue.

A mandatory regression test of Rambase Framework contains the following sub-checks.

1. Check the login flow
2. Check if dashboard & menu is loaded properly
3. Check if we can change between systems and DBs without errors.
4. Load a widely used app like COA(as most of the changes that happens in the framework can be tested here)
5. Check if Context menu, keyboard navigation & filters are working properly.
6. Check breadcrumb and breadcrumb navigation.
7. Change Theme/language and test from(1 - 6) again
8. Open developer window and check console for errors
9. Open developer window and check network tab for errors/unusual activity
10. Test application **with and without** debug=true (from steps 1 to 8)
11. Change browser and test 1-10 again.

Regression Testing - Apps Team

The app team usually does a manual regression test after doing a modification to a particular component in the Rambase web application or mobile application. If the code change makes a visible feature or modification — the tester, developer, and code reviewer are responsible for the regression test. Otherwise, only the developer and code reviewer are responsible.

A typical regression test has the following flow.

1. The developer makes a change in the code by implementing a feature or fixing a bug.
2. The developer reviews the code and does a regression test.
3. If the code change is visible via the frontend, a tester will do a regression test.
4. Code reviewer does a regression test by reviewing the code and testing the feature/bugfix.
5. Modified components will be sent for deployment.

A mandatory regression test of a Rambase application contains the following sub-checks.

1. Check whether the application is loading without any issue.
2. Check parameter mappings by changing parameters. Eg: If there are two params CUS/{par1}/{par2}, try to change parameters: par1 and par2.
3. Make sure that the application works fine with the back button, escape key, and breadcrumb.
4. Inspect the console logs. Are there any new warnings or errors?
5. Check the network tab. Are there too many API calls with the parameter change?

Regression Testing - RACR

The RACR team usually does a manual regression test after doing a modification to a particular component in the RACR project. The developer and code reviewer are responsible for the regression test.

A typical regression test has the following flow.

1. The developer makes a change in the code by implementing a primitive or fixing a bug in a primitive.
2. The developer reviews the code and does a regression test.
3. Code reviewer does a regression test by reviewing the code and testing the primitive/bugfix.
4. Modified components will be sent for test in the test environment which also managed by the developers.

List of mandatory regression tests :

Modification/ Bugfix	Tests
Filterbuilder	1. FilterBuilder Primitve Tests 2. Calculate Primitive Tests 3. Exists Primtive Tests 4. TextSearch Primitive Tests 5. CreateBatchData Primitive Tests
CosCache	1. GetCosMeta Primitive Tests 2. FindCosVersion Primitive Tests 3. GetCosVersionMeta Primitive Tests
DictionaryCache	1. GetCompanyId Primitive Tests 2. GetArchiveTemplate Primitive Tests 3. GetArchiveClasses Primitive Tests 4. GetCompaniesInDBG Primitive Tests 5. GetArchiveField Primitive Tests 6. GetArchiveFields Primitive Tests 7. GetCompaniesSharingArchives Primitive Tests 8. GetArchiveCustomFields Primitive Tests
RateCache	1. GetRate Primitive Tests 2. IsValidCurrency Primitive Tests 3. GetCurrencies Primitive Tests
DynamicCosLoader	1. IsProductionSystem Primitive Tests 2. NotifyServiceBus Primitive Tests
ScanRestrictions	1. ScanApp Primitive Tests 2. ScanAcs Primitive Tests 3. ScanForm Primitive Tests 4. ScanScv Primitive Tests

Regression Testing - Output Team

Output team is working on several sub-projects under these two categories

Rambase Printing Solution

Allows users to print their invoices reports and other rambase related documents via remotely connected printing devices.

Print Cloud - Print Server

1. Manual regression test carried out by the developer and deploy the test environment
2. Deploy through Azure DevOps - partially integrated automated test case execution when deploying to the test environment
3. Product owner carried out manual testing and regression test using customer's QA resources and approves to production deployment
4. Product owner approves and triggers production deployment

Print Connector - Client-side windows Service

1. Manual regression tests carried out by the developer and manual deployment of executables as a test print connector
2. The product owner verifies the print connector and makes it available to the production environment

A mandatory regression test of the Rambase Printing solution contains the following sub-checks

1. SignalR/Connectivity check with print connectors and print cloud
2. Manual print trigger using Printcloudtest app (directly executes the print job)
    a. Testing non-collated prints
    b. Testing collated print
3. Trigger print using Output Queue (which is the default flow to execute the print event in Rambase) and verity the print event received to the print queue
4. Verify the print statuses updated properly on the Rambase side for both collated and non-collated prints.

Rambase Shipping Integration

Integrate third party shipping services to Rambase such as Consignor, Unifaun, Timpex etc.

Ship shipment API plugin - the connector of Shipment API to Rambase

1. Manual regression testing before production deployments.
2. The Product Owner makes available the Plugin to the production after testing is carried out.

Shipment API - map different shipping services to rambase

1. Manual regression test carried out by the developer and deploy the test environment
2. Deploy through Azure DevOps - partially integrated automated test case execution when deploying to the test environment
3. Product owner carried out manual testing and regression test using customer's QA resources and approves to production deployment
4. Product owner approves and triggers production deployment

A mandatory regression test of Rambase Shipping Integration prior to a deployment contains the following sub-checks

1. Change Shipping Service credentials to their sandboxes
2. Carry out shipment without packages for each carrier
3. Make shipments including packages and validate the package details and their validations (display proper validations according to the shipping service).
4. Monitor the statuses that are properly syncing with the broker portal. (open, shipped etc)
5. Verify the reporting webhooks are triggered properly and auto-generate/ download the necessary documents with the expected template.
6. Verify the parameter conversions and dimensions are passed properly to the broker portal.
7. Update the credentials with production credentials.

Discussion Suggestions

Meeting (3rd of June 2021)

Participants:

@ Sachith Jayasinghe @ Hashan Wanniarachchi @ Chaminda Vithanage

	Suggestion	Description	Status
1		So its easy to monitor the team performance as well as identify the technical/domain gaps within the team.	Todo

	Firgureout a way to create a report based on estimate vs actual time for assigned tasks	Better approach is setting storypoints. Need to come up with a plan to pitch it to customer.	
2	Develop a practice to comment on the current task if you are switching to a critical issue when its in “in-progress”	Provide more tracebility to the task with respect to created date, spent time and completed date.	Todo
3	Verify the GDPR module completed by the all team members. Gather related evidence (agreement with customer etc)	Obtain sound knowledge about GDPR compliences within the team and make sure there’s no personal identifiable information access or do it in a proper manner if so.	Todo
4	Internal Team Satisfaction plan	maintaining a proper internal team satisfaction procedures within the team.	Todo
5	(CDP) Countinuous Development Plan	Implement a CDP along with a Succession Plan (Make sure there’s succeser to play your role). Properly document responsibility deligation and trainings.	Todo
6	Quality Review of team members	Have a monthly evaluation on team members' non-project (Innovative related) engagement of activities.	Todo
7	Issue Escalation Plan	Maintain a list of Resource people (SL & Norway) and their contact details for each domain so its easy to contact them directly.	Todo
8	Create a Access Privilage Record	Maintain a document of who has access to what systems	Todo

Issue Escalation Plan

Resource persons (SL & Norway)

Team	Issue	Resource Person
Framework Team	Core feature functionality issues	• Samudra Kanankearachchi • Hashan Wanniarachchi
	Core feature usability/styling issues	• Raashid Ahamed
	Login issues, Access issues, Permission issues	• Ruben Olsen
UX/UI	User Experience & Usability issues in an application	• Roger Gullhaug • Kjell-johan • Raashid Ahamed
	User-Interface related issues	• Raashid Ahamed
App Team	CRM/HR/Rental apps and Mobile app related	• Kamil Polikiewicz
	COS programming	• Roald Årvik
	API design/approval	• Bjørn Petter Nilsen
	Batch jobs	• Odd Egil Dvergsnes
Print Team	The print solution, Print cloud, Algorithm, Microservices, print connector, output repository, Telerik runtime, shipping integration	• Sachith Sujeewa
	Output related COSes, domain details	• Ragnar Wiik Johansen
	Design Architectural decisions	• Ruben Haakonseth Olsen
RACR Team	APIs related issues / Rabbit related issues (new RACR COSes, profiler)	• Bjørn Petter Nilsen
	DAD related issues/ Engine/ Transpiler / VScode plugin	• Håkon Andrè Førre Knudsen • Kalpani Ranasinghe - VScode plugin
	Database access related issues	• Jens Hittenkofer

	CLI related issues	• Håkon Andrè Førre Knudsen • Kalpani Ranasinghe
	Async update in Primitives / SQL queries	• Alf Inge Hovland
	Primitives	• Alf Inge Hovland • Kalpani Ranasinghe • Håkon Andrè Førre Knudsen (Cos Table primitives) • Bjørn Petter Nilsen (Primitive Interfaces)
	Caches	• Jens Hittenkofer (Permission, AruAop, Dictionary) • Kalpani Ranasinghe (Rate, Settings, Cos)
Other Issues	VPN issues	• EMP Secure team
	Rb-Homer isssues	• Are Leland
	Rabbit issues	• Bjørn Petter Nilsen

Career Development Plan

https://docs.google.com/presentation/d/1hHW3eVZjG-qWMl1Jl3X3NWYp16efMjAeKY12LTmSBCI/edit#slide=id.p - Career Development Plan presentation

Example CDP Template

Srilal Siriwardhana

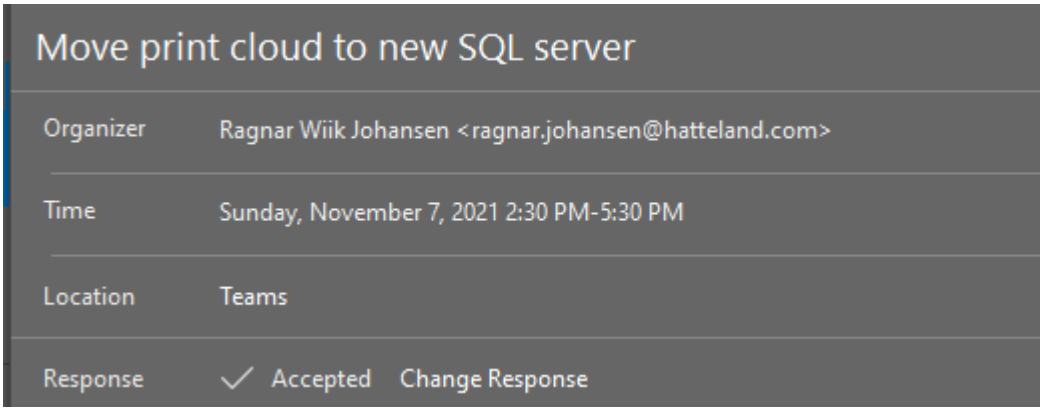
The career development plans of @ SrilalS will be under this subsection

Architectural decision log

Date	Subteam	Decision taken	Context		Acknowledgment

				Any supporting documents	
2020/04/04	Framework Team	.Net Core Technology migration (Owner Samudra kanankearachchi)	Client Framework / Rambase Web Application	Initial prototype was done using .Net Core , after proven successful entire product has being migrated to .net core tested and now on production. www.rambase.net	Acceptance trough regular demos and QA testing and End user testing . Now it is running in production more than 6 months .
2021/5/01	Framework Team	Investigate / Prototype file upload service to support large file uploads.	Rambase File Upload Service	Prototype is ongoing with chunk upload implementation .	

Production environment access details

Date and time	Subteam	Is GDPR related content accessed?	Context	Acknowledgement
07-11-2021 2.30 PM - 5.30 PM	Output	No	Production Database Migration to new SQL server	

Security Testing Specification

Introduction

The development team typically identifies vulnerabilities in codebases during code review sessions. Code review sessions are occurring every two weeks, and the development team often does security tests for selected modules during code review meetings. Also, the dedicated security testing team from the customer's side reports critical security vulnerabilities to responsible development teams.

For internal security testing (in code reviews), the development team uses the following security checklist. This checklist is common for all developers who work with application components, API modules, print servers, development tools, and mobile applications.

Security Testing Checklist For Developers

- If you are rendering/storing user-entered HTML content, make sure that the HTML content is properly sanitized.
- Avoid extracting and using the access token explicitly in application components - instead, let the client framework securely handle it.
- Do necessary in-depth security testing for places where WYSIWYG editors are used and identify possible XSS vulnerabilities.
- Make sure that the API modules use correct permissions (PRM) and access rules (ARU).
- Make sure to expose data records according to permissions from the API.
- Test batch jobs for different user inputs and make sure that those user inputs are not causing unwanted data manipulations.
- Test token expiry and validity for code changes in modules that implement auth flows such as mobile app, client framework, and print module.
- Make sure that the code implements necessary validations for user inputs that are used for internal data manipulations, calculations, and access control.

Advanced Security Testing Sessions

Sometimes, quick code review sessions are not enough to do extended in-depth security tests for components that should be thoroughly tested in security aspects. In those scenarios, use OWASP Testing Guide v4.2 as an extension to the above general checklist.

Action Items

Once your team detects a security breach or vulnerability, create and prioritize a task on the Jira board with a descriptive title and description.

Eg:

https://rambase.atlassian.net/browse/CRM-1785

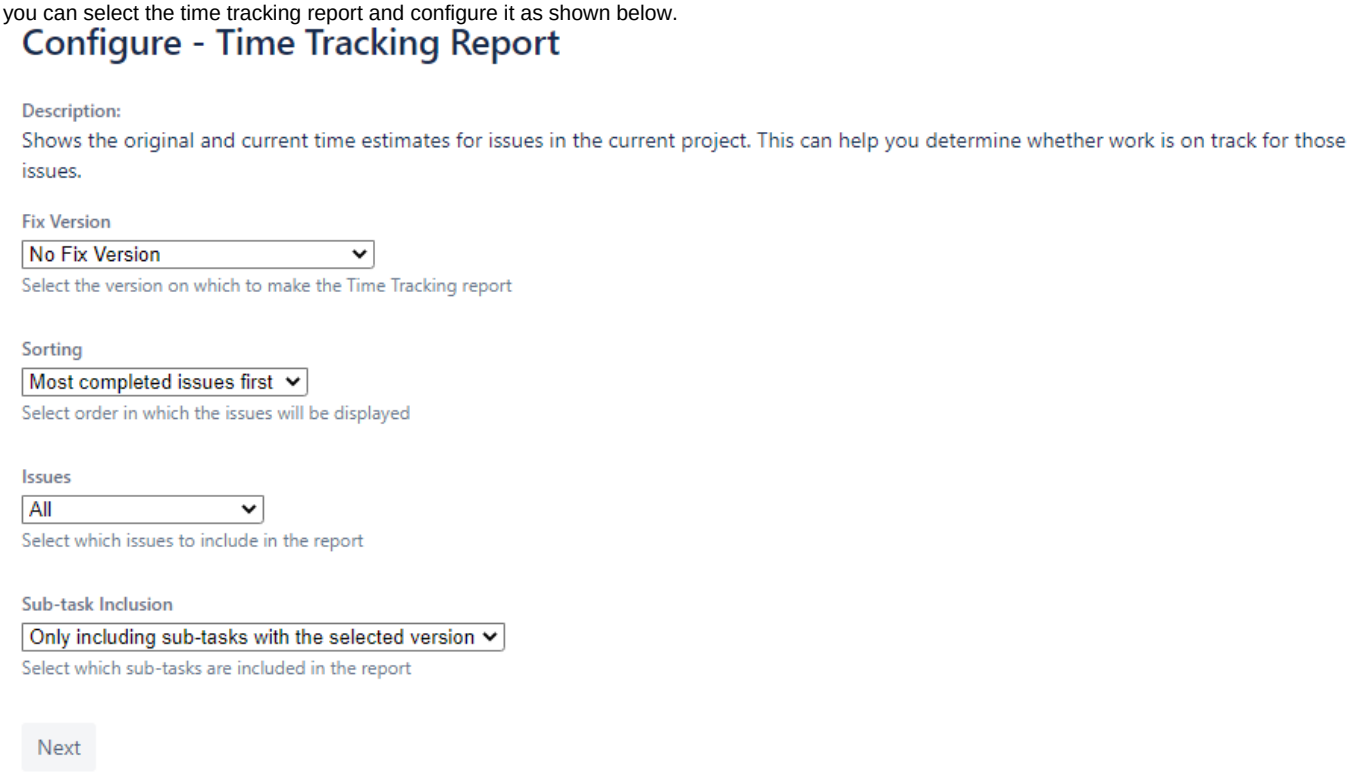
Quality Metrics

Time Tracking Report

Shows the original and current time estimates for issues in the current project. This can help you determine whether work is on track for those issues.

Under the forecast and management,

you can select the time tracking report and configure it as shown below.



Generated report

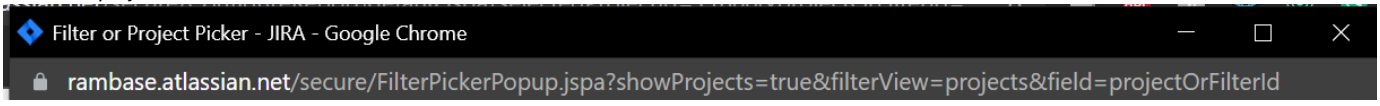
Key		Summary	Original Estimate	Σ	Est. Time Remaining	Σ	Time Spent	Σ	Accuracy	Σ
RPS-3360	CLOSED	RPS-3191 Create the norwegian translation and test design	2h	2h	0h	0h	1.5h	1.5h	0.5h	0.5h
RPS-3361	CLOSED	RPS-3194 create new design and test generation of the translation file	4h	4h	0h	0h	4h	4h	on track	on track
RPS-3362	CLOSED	RPS-3194 Create the norwegian translation and test design	2h	2h	0h	0h	2h	2h	on track	on track
RPS-3363	CLOSED	Identify and replace text in for all design using text "Number of packages...	0h	0h	0h	0h	4h	4h	-4h	-4h
RPS-3364	CLOSED	In OOS and OPD application it seems as every change we do makes it ha...	12h	12h	0h	0h	20h	20h	-8h	-8h
RPS-3365	CLOSED	RPS-3153 Create the design	0h	0h	0h	0h	5h	5h	-5h	-5h
RPS-3366	CLOSED	RPS-3153 Test translation	0h	0h	0h	0h	1h	1h	-1h	-1h
RPS-3368	CLOSED	Move some field for the Shipping advice / Packing slip (CSA) in the gene...	4h	4h	0h	0h	4h	4h	on track	on track
RPS-3369	CLOSED	Do a final test new output designs listed in description	12h	12h	0h	0h	10h	10h	2h	2h
RPS-3370	CLOSED	Move the invoice date above the due date on all sales invoice designs	3h	3h	0h	0h	3h	3h	on track	on track
RPS-3373	CLOSED	Add support for multiple tables inside lists and panels.	0h	0h	0h	0h	4h	4h	-4h	-4h
RPS-3376	CLOSED	Create the specifications for rental and CSO output designs	5h	5h	0h	0h	4h	4h	1h	1h
RPS-3377	CLOSED	Add "BILL TO" all SPO designs and the same way as for CSA documents	0h	0h	0h	0h	7h	7h	-7h	-7h
RPS-3379	CLOSED	For all SPO documents with price add Document.Currency behind price a...	0h	0h	0h	0h	4h	4h	-4h	-4h

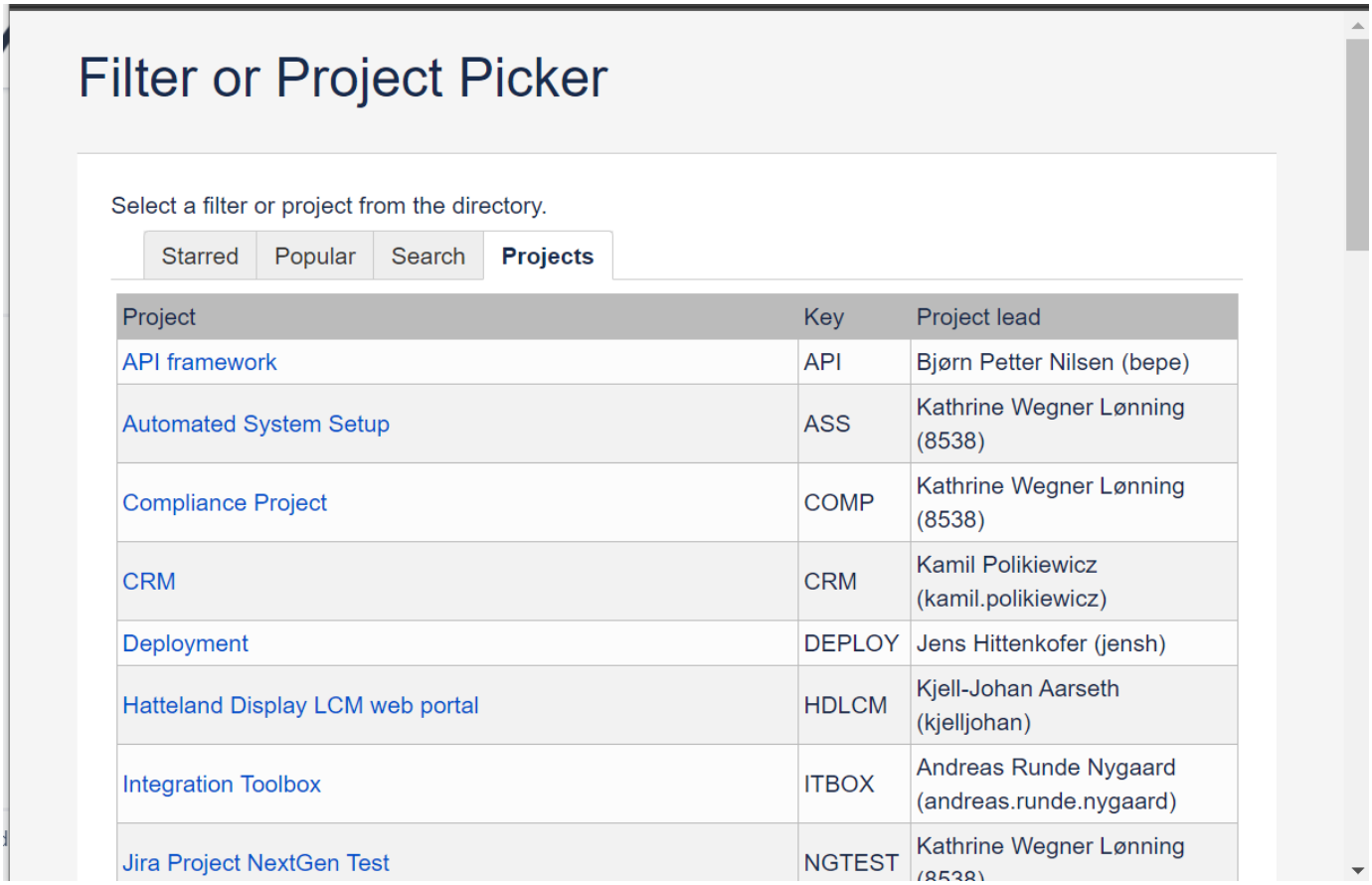
These are the reports that can be generated from Jira related to issue tracking and quality assurance of the product

Configure - Created vs. Resolved Issues Report

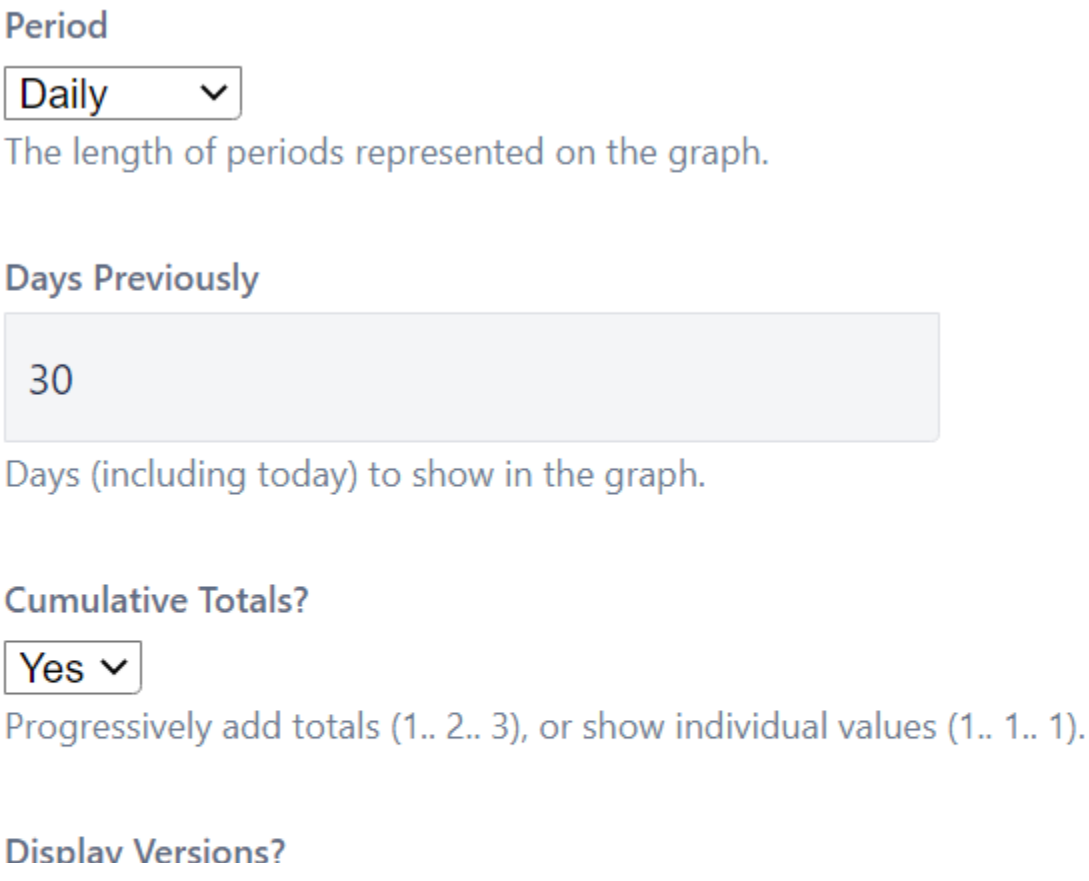
Description:
Above report maps created issues versus resolved issues over a period of time. This can help you understand whether your overall backlog is growing or shrinking.

select the project

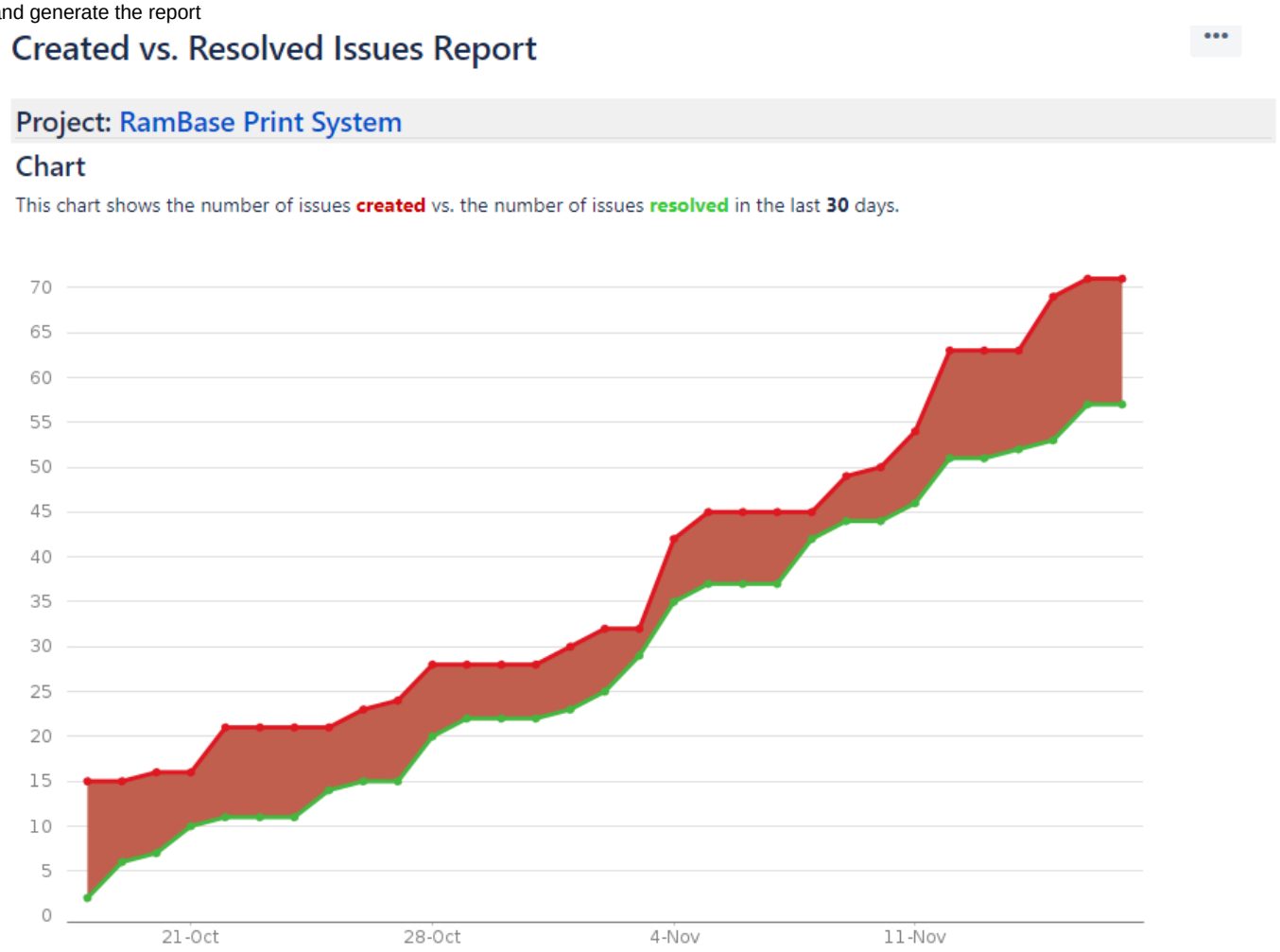




Set the below criteria as needed



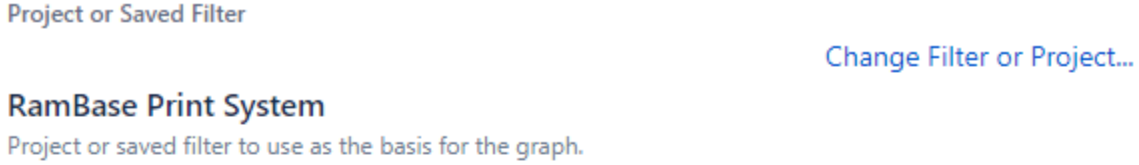
and generate the report



Configure - Recently Created Issues Report

Description:
Shows the number of issues created over a period of time for a project/filter, and how many were resolved. This helps you understand if your team is keeping up with incoming work.

Select the project and chose the parameters



Period

Daily

The length of periods represented on the graph.

Days Previously

30

Days (including today) to show in the graph.

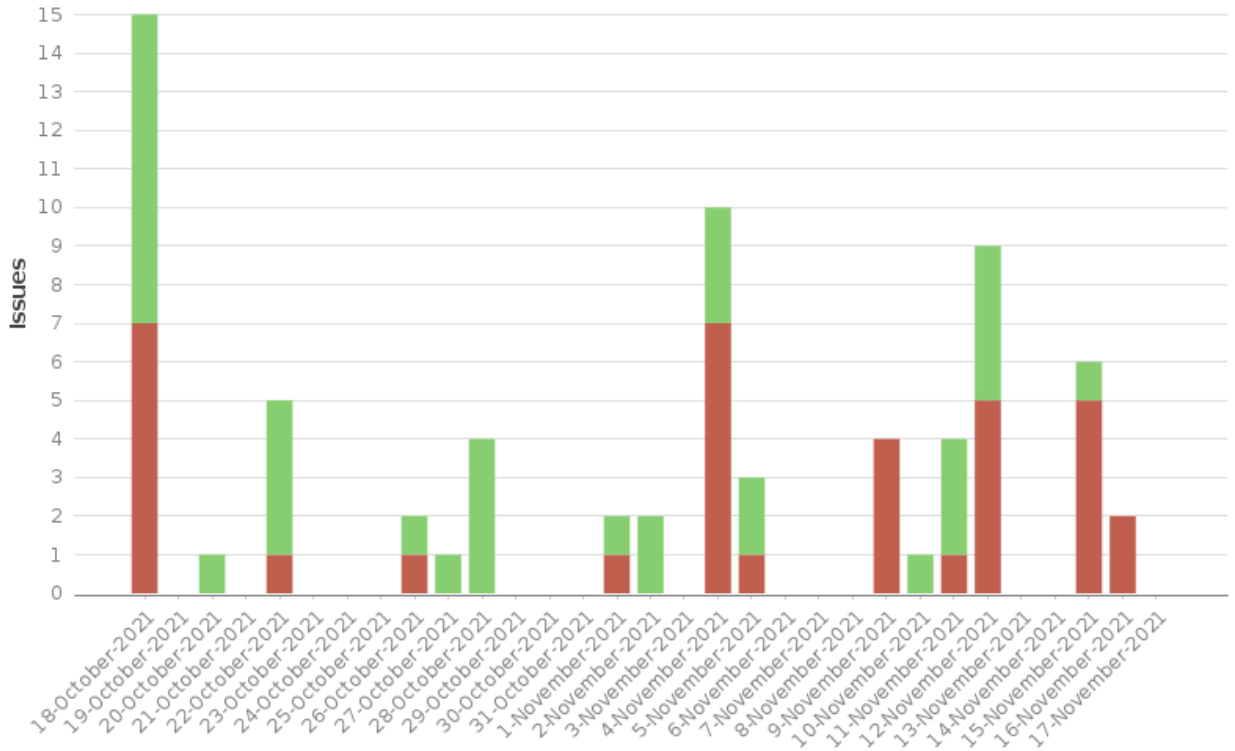
Next

Recently Created Issues Report

Project: RamBase Print System

Chart

This chart shows the issues created in the last **30** days



Checklist based Test Scenarios

Sub development teams use a generic checklist-based testing guideline to verify the quality of deliverable tasks. Also, each sub-team needs to do a team-specific mandatory regression test along with the generic checklist-based test scenario. Every developer uses the following generic checklist to verify the deliverable quality.

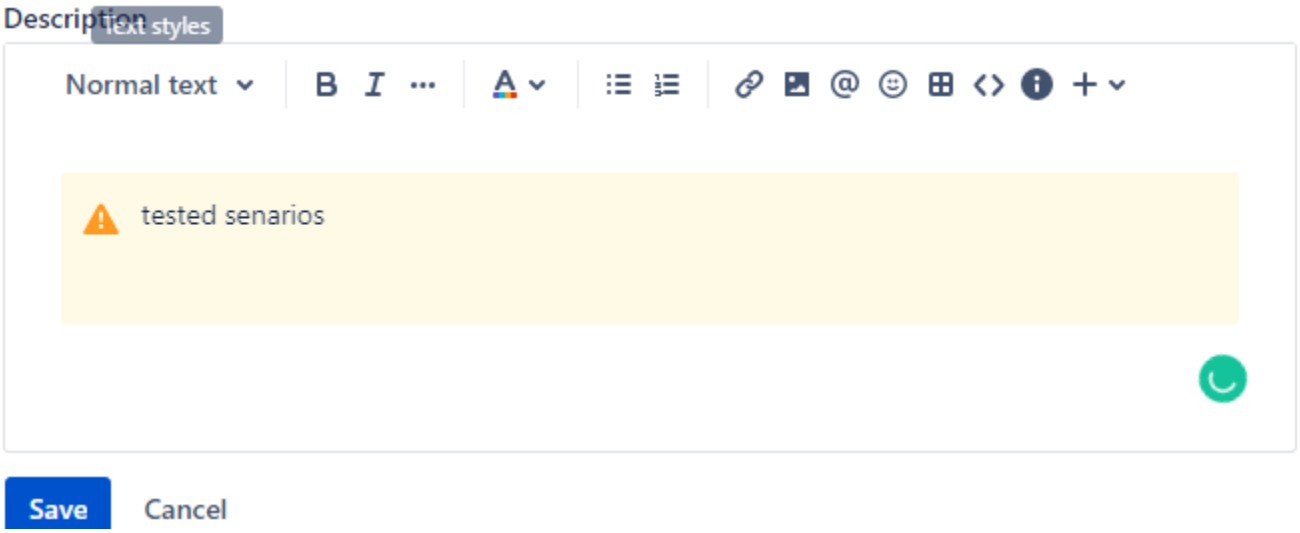
- Make sure that your code's logical flow doesn't introduce undefined situations. For example, if your code only accepts input X, you need to properly handle your code's behavior for input Y.

- Test the new feature with invalid user inputs, long user inputs, and inputs with special characters to verify error handling.
- Check whether the newly added UI elements adhere to Rambase UI development guidelines.
- Make sure that the application renders well on different screen sizes and slow network conditions. (Use Chrome's device emulation)

The above testing checklist is mainly for developers to verify deliverable quality. But, it is always great if the reviewer also checks the above points again. The development team motivates testers and reviewers to check the following points in addition to the above checklist points.

- Verify the implementation with the original requirement and business logic.
- If the developer added the task to the "Code Review" list, initiate a code review before the testing process.
- If the feature is multilingual, test with different languages.
- Test the usability factors of the feature by using the new feature in different possible ways. Eg: if the feature adds a new popover to the app, try opening it via keyboard shortcuts.
- Make sure that each action and error handling flow followers general development guidelines. Eg: Throws errors for missing permissions.

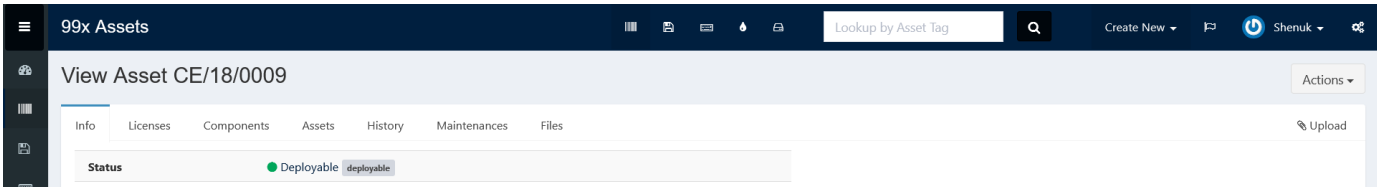
In most scenarios, all the above checklist points are not applicable for every development task. For example, implementing a backend component doesn't require frontend-related testing. Therefore, make sure to add a note about tested scenarios to the related Jira task, as shown below:

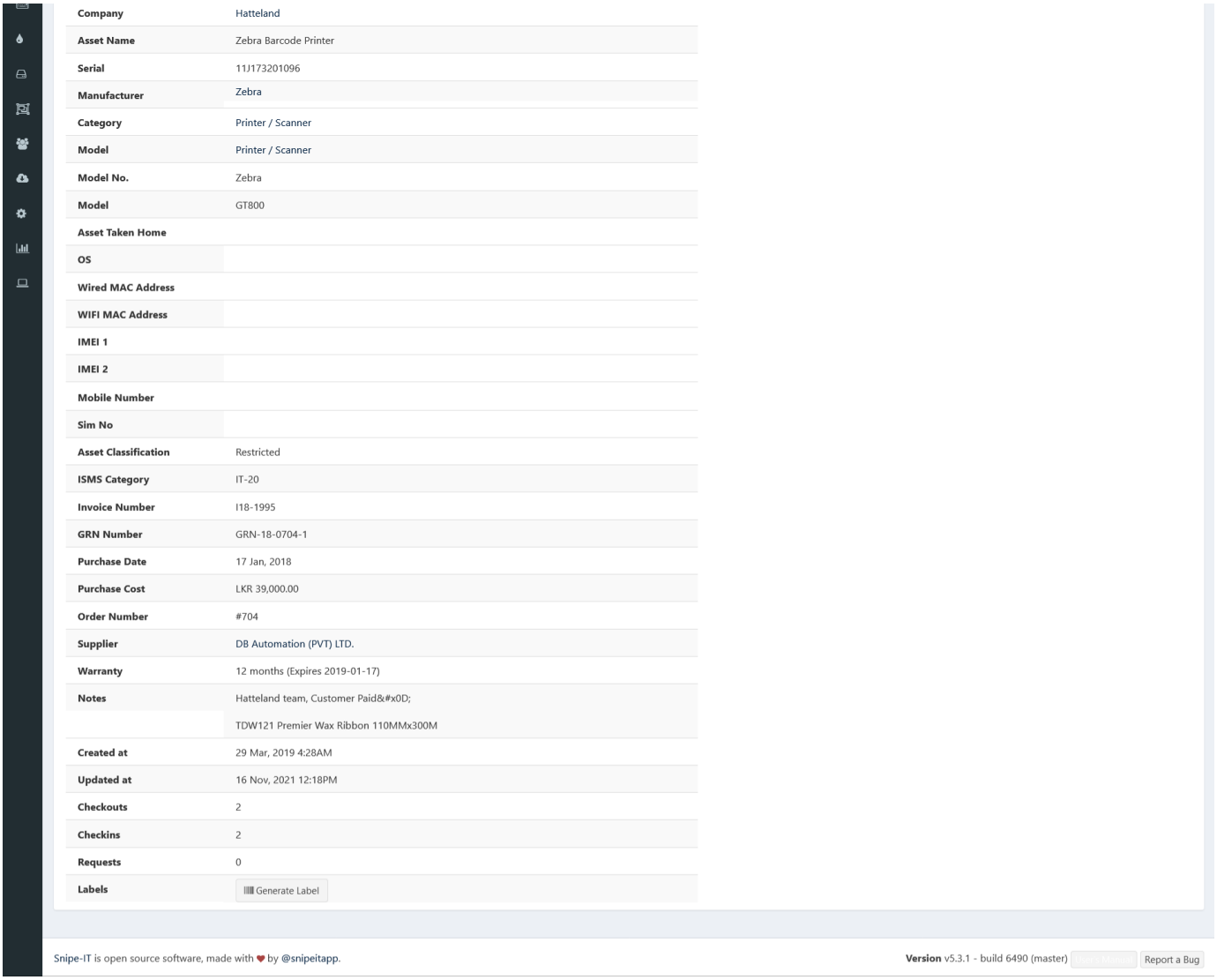


Test Device Management

1. Zebra Barcode Printer - CE/18/0009

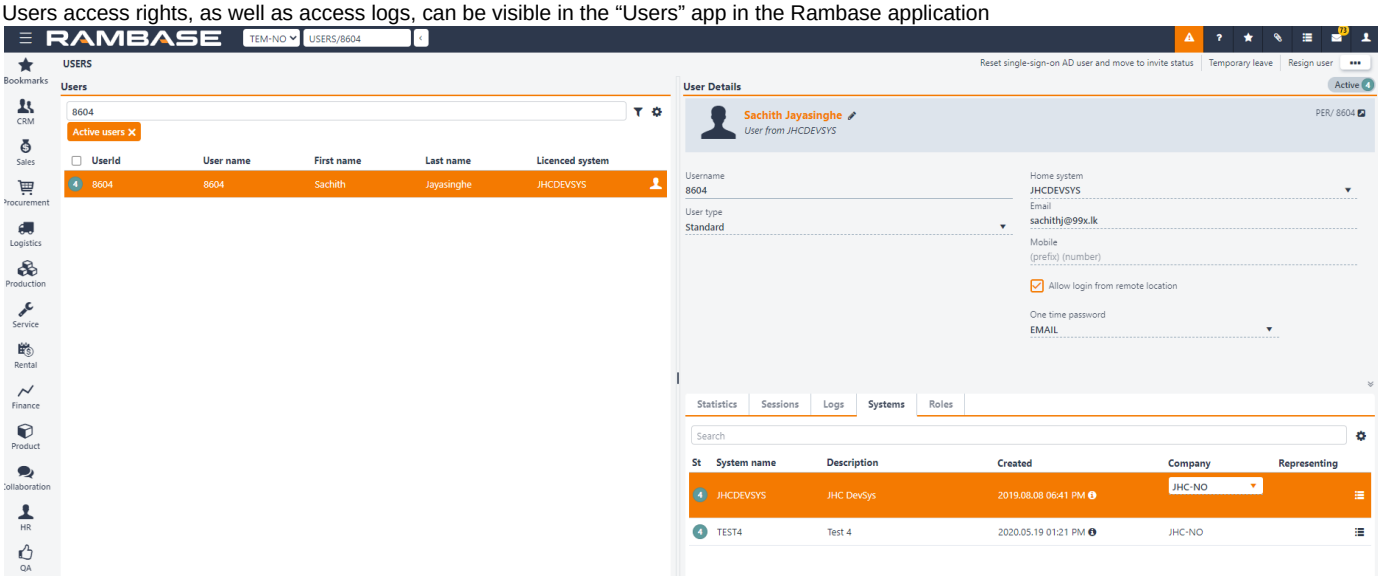
Relevant documentation is attached below.





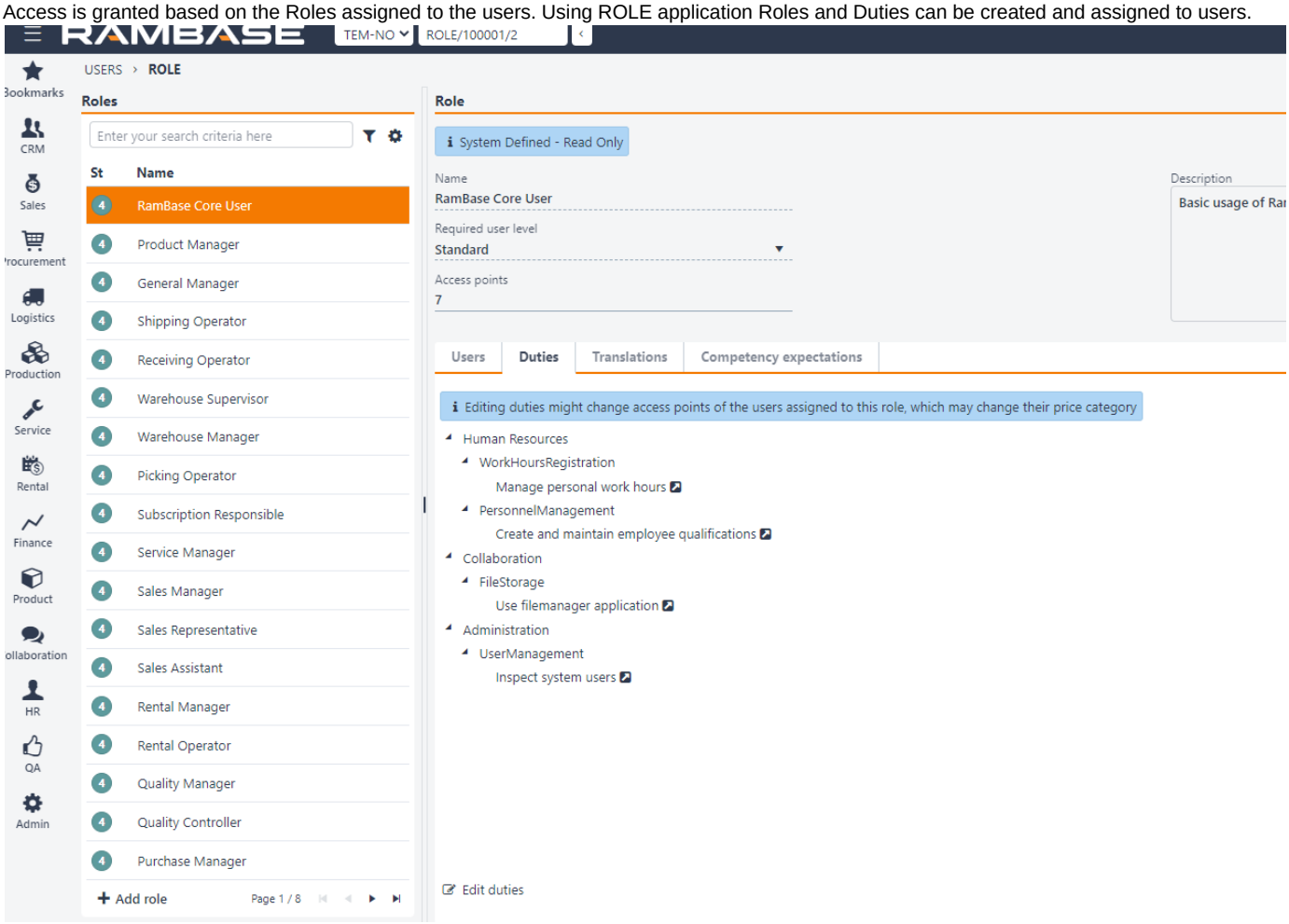
Team's Access Rights

Users access rights, as well as access logs, can be visible in the “Users” app in the Rambase application



Sessions Logs and access systems can be visible from here.

Access is granted based on the Roles assigned to the users. Using ROLE application Roles and Duties can be created and assigned to users.



2022 - H2

1676 – Hashan

1674 – Hashan

1675 – Hashan/Sachith

1673 – Savindu/ Warsha

1615 – Obhasha/ Sachith

1614 – Sahan/Sachith

1613 - Fazna

Information Security Principals

Plan to implement the defined information security principles shall be developed (Refer work instruction “200WI01 Secure System Engineering Principles”)

Status
Pending

Labels
Engineering Discipline
Complience
Architecture & Dev

Assesment Type
Product Assesment

Update discipline

Engineering Discipline: CM.1.3

Required information security principalns identified and planned for implementation [ISO]

Detailed description
[ISO 27001 A.14.2.5] Plan to implement the defined information security principles shall be developed (Refer work instruction “200WI01 Secure System Engineering Principles”)

Compliance:

1.1 Purpose

Purpose of this work instruction is to provide a guideline for the project teams in implementing information security and privacy principals in software engineering.

1.2 References

1.2.1 ISO 27001:2013

1.2.2 ISO 27701:2019

1.3 Responsibility

1.3.1 Team Lead

1.4 Activities

1.4.1 Secure System Engineering Principals

1.4.1.1 Customer’s security goals shall be identified and documented during the application design stage.

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/3353640961/Rambase+Security+-+Q+A

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3099328523
/Annual+information+security+audit+of+our+sub+contractors+99x+and+Fabres

https://rambase.atlassian.net/wiki/spaces/RDTL/pages/2850825/RamBase+security+-+In+the+context+of+the+API

https://rambase.atlassian.net/wiki/spaces/API/pages/732430371/API+Server+security

https://rambase.atlassian.net/wiki/spaces/RLS/pages/340197392/RamBase+login+Security+and+Authentication

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3144679430/Procedure+for+handling+information+security+incidents

1.4.1.2 Information security and privacy shall be considered as an integral part of system design and shall ensure the identified security goals are met in it.

Enterprise Privacy Architecture

1.4.1.3 A cost benefit analysis should be done for identified security and privacy goals to formulate a mitigation strategy for them. It’s recognized that elimination of all security and privacy risks is not cost effective.

1.4.1.4 Developers should be trained on developing secure software.

1.4.1.5 All 3rd party systems/services should be treated to be insecure until they are deemed trusted by conducting a security audit or signing of a secure service agreement.

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/1146421264/Third-party+Service+Providers

1.4.1.6 Potential trade-offs should be identified between reducing security and privacy risk and increased costs and decrease in other aspects of operational effectiveness.

1.4.1.7 Tailored system security and privacy measures should be implemented to meet organizational security and privacy goals. In case of adopting a common security measure, it’s adequate to cover the set security and privacy goals should be ensured.

1.4.1.8 Information, specially PII shall be protected while being processed, in transit, and in storage.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment

https://rambase.atlassian.net/wiki/spaces/PROC/pages/297861606/Guidelines+for+Customer+data+handling

1.4.1.9 Relevant classes of attacks should be identified according to the set security and privacy goals. Examples of “attack” classes are: Passive monitoring, active network attacks, exploitation by insiders, attacks requiring physical access or proximity, insertion of backdoors and malicious code during software development and/or distribution.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3144679430/Procedure+for+handling+information+security+incidents

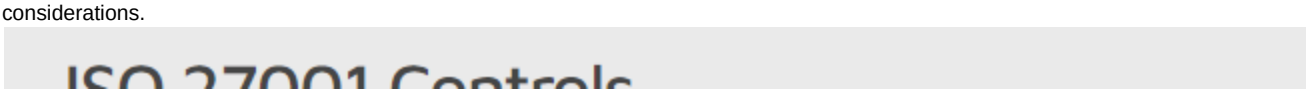
1.4.1.10 Secure application design practices should be incorporated to ensure interoperability and portability into all security measures, including hardware and software, and implementation practices.

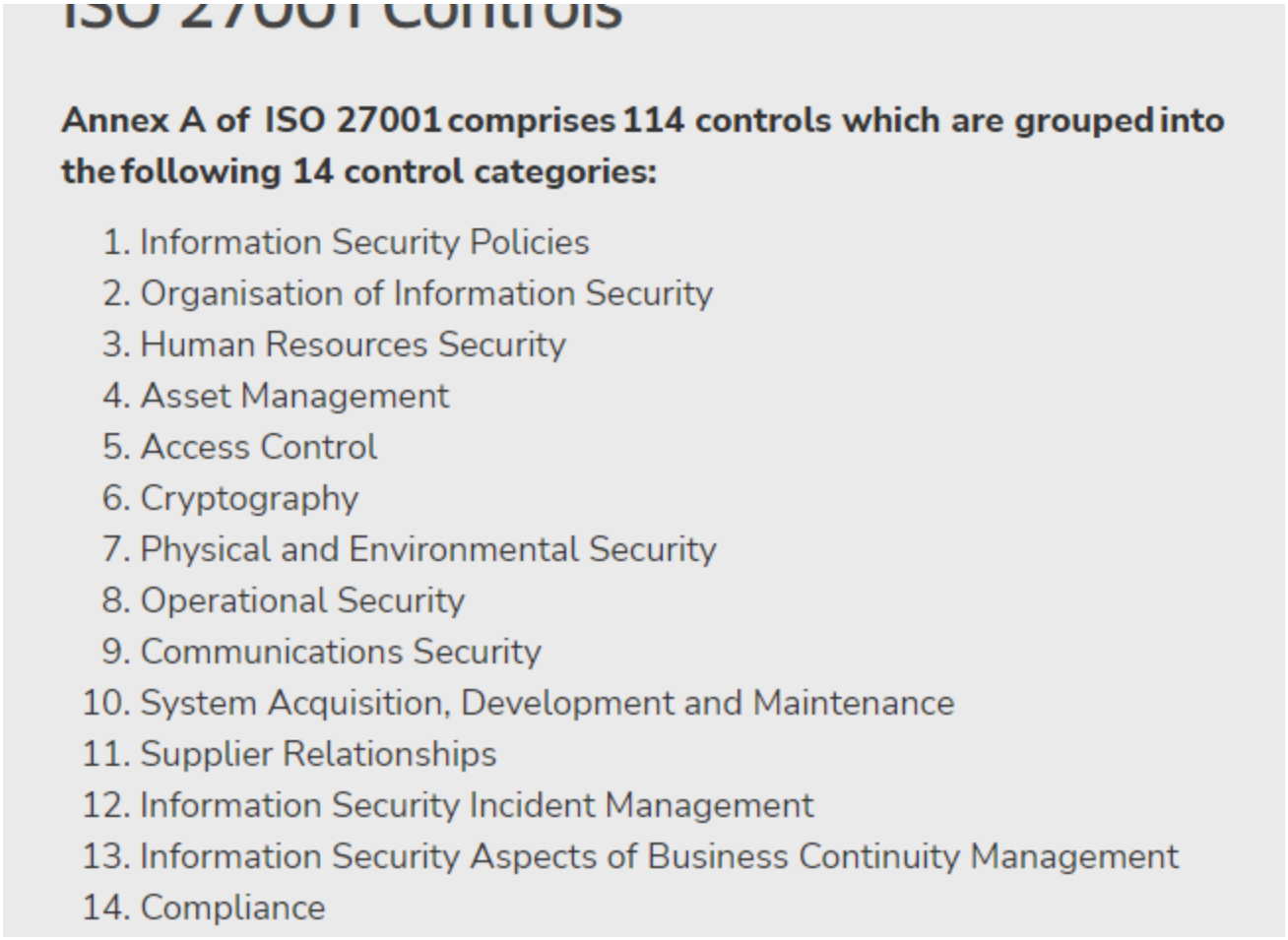
Enterprise Privacy Architecture

1.4.1.11 Applications should be designed to allow regular adoption of new technology, including a secure and logical technology upgrade processes.

Enterprise Privacy Architecture

1.4.1.12 Security controls should be designed to be consistent with the concept of operations and with ease-of-use as an important considerations.





find matching resources from customer confluence

1.4.1.13 Layered security should be implemented to ensure not having a single point of vulnerability.

Enterprise Privacy Architecture

1.4.1.14 Information systems should be resistant to attack, should limit damage, and should recover rapidly when attacks do occur. The principles suggested here recognizes the need for adequate protection technologies at all levels to ensure that any potential cyber-attack will be countered effectively. There are vulnerabilities that cannot be fixed, those that have not yet been fixed, those that are not known, and those that could be fixed but are not (e.g., risky services allowed through firewalls) to allow increased operational capabilities. In addition to achieving a secure initial state, secure systems should have a well-defined status after failure, either to a secure failure state or via a recovery procedure to a known secure state. Organizations should establish detect and respond capabilities, manage single points of failure in their systems, and implement a reporting and response strategy.

Enterprise Privacy Architecture

Fault tolerance, resilience, defense in depth, containment, etc. explanation

1.4.1.15 It should provide assurance that the system is, and continues to be, resilient in the face of expected threats. Continuous security analysis and vulnerability scanning should be implemented.

Enterprise Privacy Architecture

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/336396289/Operations+Change+Plan

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3099328523
/Annual+information+security+audit+of+our+sub+contractors+99x+and+Fabres

1.4.1.16 Systems design should be done to limit or contain vulnerabilities. If a vulnerability does exist, damage can be limited or contained, allowing other information system elements to function properly. Limiting and containing insecurities also helps to focus response and reconstitution efforts to information system areas most in need.

Enterprise Privacy Architecture

1.4.1.17 Boundary mechanisms should be used to separate computing systems and network infrastructures. (eg: segregate development, testing and production environments)

Enterprise Privacy Architecture

Add DMZ and other boundaries in a diagram

1.4.1.18 Audit mechanisms should be designed and implemented to detect unauthorized use and to support incident investigations.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3099328523
/Annual+information+security+audit+of+our+sub+contractors+99x+and+Fabres

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3144679430/Procedure+for+handling+information+security+incidents

1.4.1.19 Contingency or disaster recovery procedures should be designed to ensure appropriate availability.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/3144679430/Procedure+for+handling+information+security+incidents

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/254869766/Disaster+Recovery+Procedures

https://rambase.atlassian.net/wiki/spaces/RBOP/overview

1.4.1.20 Simplicity should be considered when implementing security measures. The more complex the mechanism, the more likely it may possess exploitable flaws. Simple mechanisms tend to have fewer exploitable flaws and require less maintenance. Further, because configuration management issues are simplified, updating or replacing a simple mechanism becomes a less intensive process.

Enterprise Privacy Architecture

1.4.1.21 Every security mechanism should support a security service or set of services, and every security service should support one or more security goals. Unnecessary security mechanisms should not be implemented.

Enterprise Privacy Architecture

identified security goals should be included here

1.4.1.22 Proper security should be ensured in the shutdown or disposal of a system. At the end of a system's life-cycle, system designers should develop procedures to dispose of an information system's assets in a proper and secure fashion.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/2106556436/Ending+a+customer+RamBase+system

https://rambase.atlassian.net/wiki/spaces/PROC/pages/72122429/Termination+of+employment+-+closing+the+PID

1.4.1.23 Measures should be implemented to prevent common errors and vulnerabilities such as buffer overflows, race conditions, format string errors, failing to check input for validity, and programs being given excessive privileges.

Secure Coding trainings will handle this

1.4.1.24 Security measures should be formulated to address multiple overlapping information domains. An efficient and cost effective security capability should be able to enforce multiple security policies to protect multiple information domains without the need to separate physically the information and respective information systems processing the data. This principle argues for moving away from the traditional practice of creating separate LANs and infrastructures for various sensitivity levels (e.g., security classification or business function such as proposal development) and moving toward solutions that enable the use of common, shared, infrastructures with appropriate protections at the operating system, application, and workstation level.

https://rambase.atlassian.net/wiki/spaces/RCF/pages/647856565/Client+Framework+Architecture+Documentation

Enterprise Privacy Architecture

Add multitenant support related info to EPA

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/3353640961/Rambase+Security+-+Q+A#Sikker-IKT-infrastruktur

https://rambase.atlassian.net/wiki/spaces/RLS/pages/1206353921/Single+sign-on+sso+implementation+in+RamBase

1.4.1.25 Users and processes shall be authenticated to ensure appropriate access control decisions both within and across domains.

https://rambase.atlassian.net/wiki/spaces/RLS/pages/1206353921/Single+sign-on+sso+implementation+in+RamBase

1.4.1.26 Unique identities shall be used to ensure accountability. An identity may represent an actual user or a process with its own identity, e.g., a program making a remote access.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/2098954257/Portal+users+in+Freshdesk

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/315555866/VPN+users+on+realm+5010JHC

https://rambase.atlassian.net/wiki/spaces/RBOP/pages/839450641/External+VPN+Users

1.4.1.27 For the systems which process PII, Privacy by design and privay by default approach should be implemented.

Privacy Architecture documentation should be updated to reflect this

https://rambase.atlassian.net/wiki/spaces/RBBASE/pages/153171707/Roles+duties+and+permissions+v2

https://rambase.atlassian.net/wiki/spaces/RBBASE/pages/315523181/Roles+-+Access+based+price+model+for+RamBase+user+subscriptions

https://rambase.atlassian.net/wiki/spaces/ID/pages/3194650635
/Permissions+duties+and+roles+for+automated+flows+and+automated+flow+templates

https://rambase.atlassian.net/wiki/spaces/RQM/pages/2669412364/Quality+QHSES+module+-+Roles+permissions+and+duties

https://help.rambase.net/en/administration.html

1.4.2 Privacy Principals

1.4.2.1 Lawfulness, fairness and transparency

(1) You must identify valid grounds under the GDPR (known as a 'lawful basis') for collecting and using personal data.

(2) You must ensure that you do not do anything with the data in breach of any other laws.

(3) You must use personal data in a way that is fair. This means you must not process the data in a way that is unduly detrimental, unexpected or misleading to the individuals concerned.

(4) You must be clear, open and honest with people from the start about how you will use their personal data.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/PROC/pages/896991351/GDPR+compliance+in+RamBase+and+Pipedrive?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://www.rambase.com/trust-center

1.4.2.2 Purpose limitation

(1) You must be clear about what your purposes for processing are from the start.

(2) You need to record your purposes as part of your documentation obligations and specify them in your privacy information for individuals.

(3) You can only use the personal data for a new purpose if either this is compatible with your original purpose, you get consent, or you have a clear obligation or function set out in law.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/896991351/GDPR+compliance+in+RamBase+and+Pipedrive?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/TEST/blog/2019/02/27/741474918/GDPR+-+Manage+consent+on+a+Contact+Person

https://help.rambase.net/en/human-resources/guide-to-personal-information-in-rambase--gdpr-user-guide-/handling-individual-rights-in-rambase.html

https://www.rambase.com/trust-center

1.4.2.3 Data minimization - You must ensure the personal data you are processing is:

(1) adequate – sufficient to properly fulfil your stated purpose;

(2) relevant – has a rational link to that purpose; and

(3) limited to what is necessary – you do not hold more than you need for that purpose.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/896991351/GDPR+compliance+in+RamBase+and+Pipedrive?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/TEST/blog/2019/02/27/741474918/GDPR+-+Manage+consent+on+a+Contact+Person

https://help.rambase.net/en/human-resources/guide-to-personal-information-in-rambase--gdpr-user-guide-/handling-individual-rights-in-rambase.html

https://www.rambase.com/trust-center

1.4.2.4 Accuracy

(1) You should take all reasonable steps to ensure the personal data you hold is not incorrect or misleading as to any matter of fact.

(2) You may need to keep the personal data updated, although this will depend on what you are using it for.

(3) If you discover that personal data is incorrect or misleading, you must take reasonable steps to correct or erase it as soon as possible.

(4) You must carefully consider any challenges to the accuracy of personal data

https://rambase.atlassian.net/wiki/spaces/PROC/pages/896991351/GDPR+compliance+in+RamBase+and+Pipedrive?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://help.rambase.net/en/human-resources/guide-to-personal-information-in-rambase--gdpr-user-guide-/handling-individual-rights-in-rambase.html

https://www.rambase.com/trust-center

1.4.2.5 Storage limitation

(1) You must not keep personal data for longer than you need it.

(2) You need to think about – and be able to justify – how long you keep personal data. This will depend on your purposes for holding the data.

(3) You need a policy setting standard retention periods wherever possible, to comply with documentation requirements.

(4) You should also periodically review the data you hold, and erase or anonymise it when you no longer need it.

(5) You must carefully consider any challenges to your retention of data. Individuals have a right to erasure if you no longer need the data.

(6) You can keep personal data for longer if you are only keeping it for public interest archiving, scientific or historical research, or statistical purposes.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/896991351/GDPR+compliance+in+RamBase+and+Pipedrive?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://rambase.atlassian.net/wiki/spaces/PROC/pages/741409144/GDPR+compliance+fulfillment?search_id=b9076bc3-6370-4b32-a388-e541b37c4fe7

https://help.rambase.net/en/human-resources/guide-to-personal-information-in-rambase--gdpr-user-guide-/handling-individual-rights-in-rambase.html

https://www.rambase.com/trust-center

1.4.2.6 Integrity and confidentiality (security) - You must ensure that you have appropriate security measures in place to protect the personal data you hold.This is the ‘integrity and confidentiality’ principle of the GDPR – also known as the security principle.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/297861606/Guidelines+for+Customer+data+handling

https://help.rambase.net/en/human-resources/guide-to-personal-information-in-rambase--gdpr-user-guide-/handling-individual-rights-in-rambase.html

https://www.rambase.com/trust-center

1.4.2.7 Accountability - The accountability principle requires you to take responsibility for what you do with personal data and how you comply with the other principles.You must have appropriate measures and records in place to be able to demonstrate your compliance.

https://rambase.atlassian.net/wiki/spaces/PROC/pages/297861606/Guidelines+for+Customer+data+handling

https://www.rambase.com/trust-center

Information security on transit

Information (including PII) on transit shall be protected from fraudulent activity, unauthorized disclosure and modification

Update discipline

Engineering Discipline: CM.1.4

Information (including PII) on transit shall be protected from fraudulent activity, unauthorized disclosure and modification [ISO]

Detailed description

[ISO 27001 A.14.1.2, ISO 27701 B.8.4.3] Information (including PII) involved in application services passing over public networks shall be protected from fraudulent activity, unauthorized disclosure and modification. This should be achieved by means of following implementation;
(a)Satisfactory identity validation mechanisms (such as authorization/cryptographic policies/document signing) shall be implemented.
(b)Confidential information shall not be available/sent through public channels (anonymous) and it shall always sent by an identified sender to an identified

receiver.

[ISO 27001 A.14.1.3] Information involved in application service transactions shall be protected to prevent incomplete transmission, mis-routing, unauthorized message alteration, unauthorized disclosure, unauthorized message duplication or replay. This should be achieved by means of following implementation;
(a)Protocols of communication should be assessed against vulnerabilities, agreed in advance and consistent throughout the transaction.
(b)Identity verification (electronic signature) mechanism should be in place.
(c)When demanded, privacy of the entities and the confidentiality of the transactions should be respected.
(d)Transaction details, logs, audit entries or any form of a trace should not be publicly accessible via the internet.

- Http/2
- FTP
- Firewalls
- Authentication , Identity validation/verification
- No confidential info
- Log management

Security mechanisms used in services passing over public networks

2fa for bitbucket

App passwords should be created for Bitbucket and Jira to connect the respective APIs to RamBase internal development applications (such as Rabbit)

Role based authorization implemented in RamBase API.

User sessions with token-based authentication.

VPN for external developers (to be within a trusted network)

User sessions expire after 8 hours in a trusted network. 20 mins in an untrusted network.

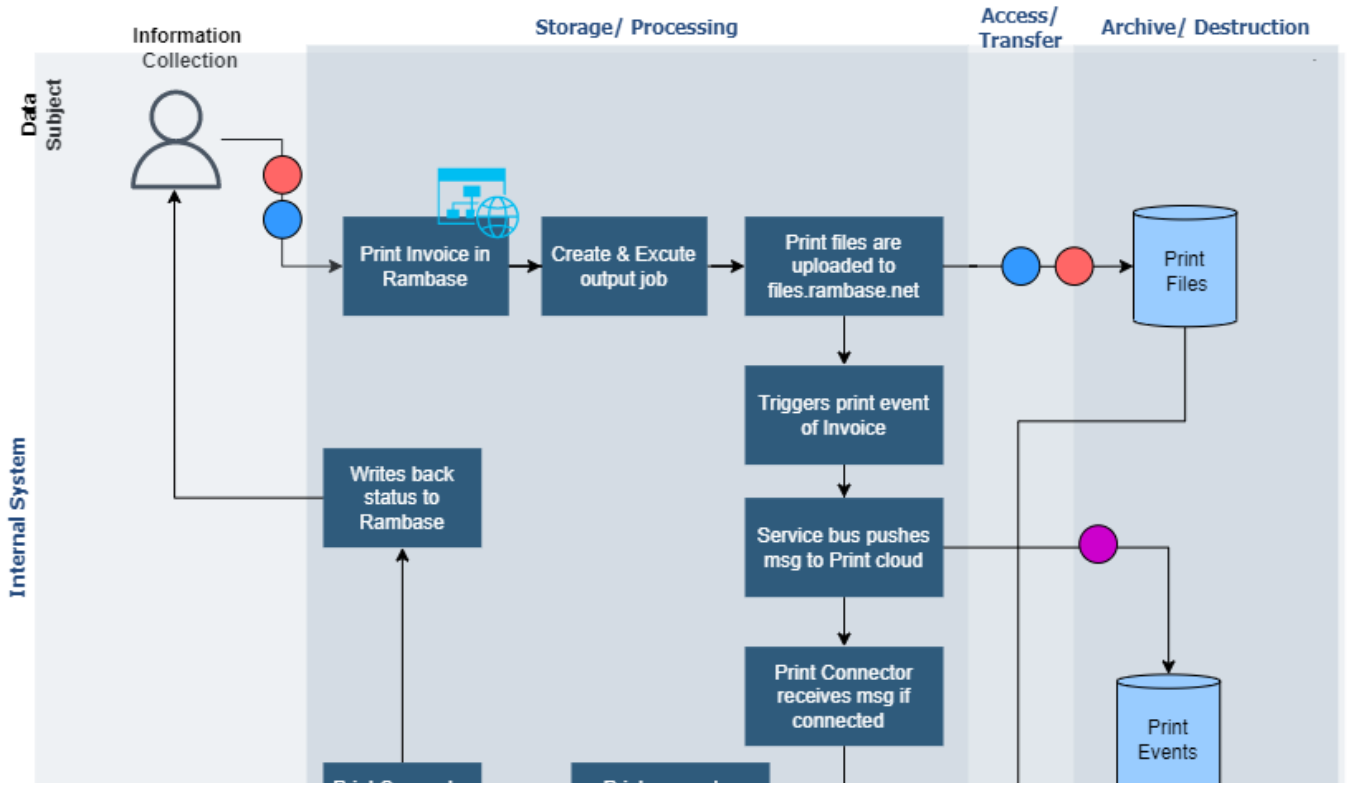
RamBase Login Security

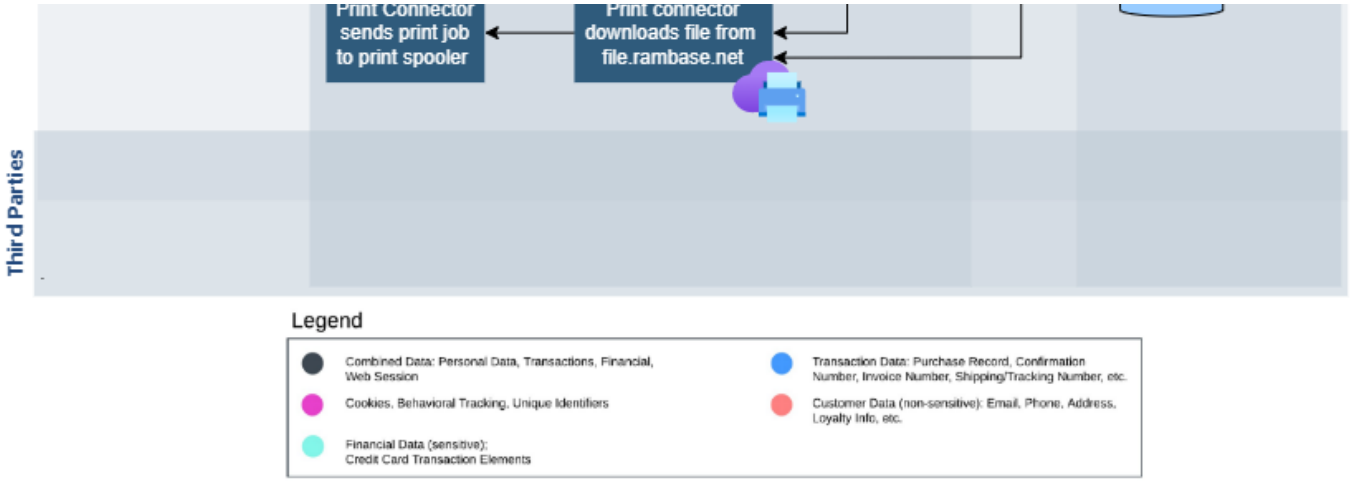
Single Sign-On (SSO) implementation in RamBase

API Server Security

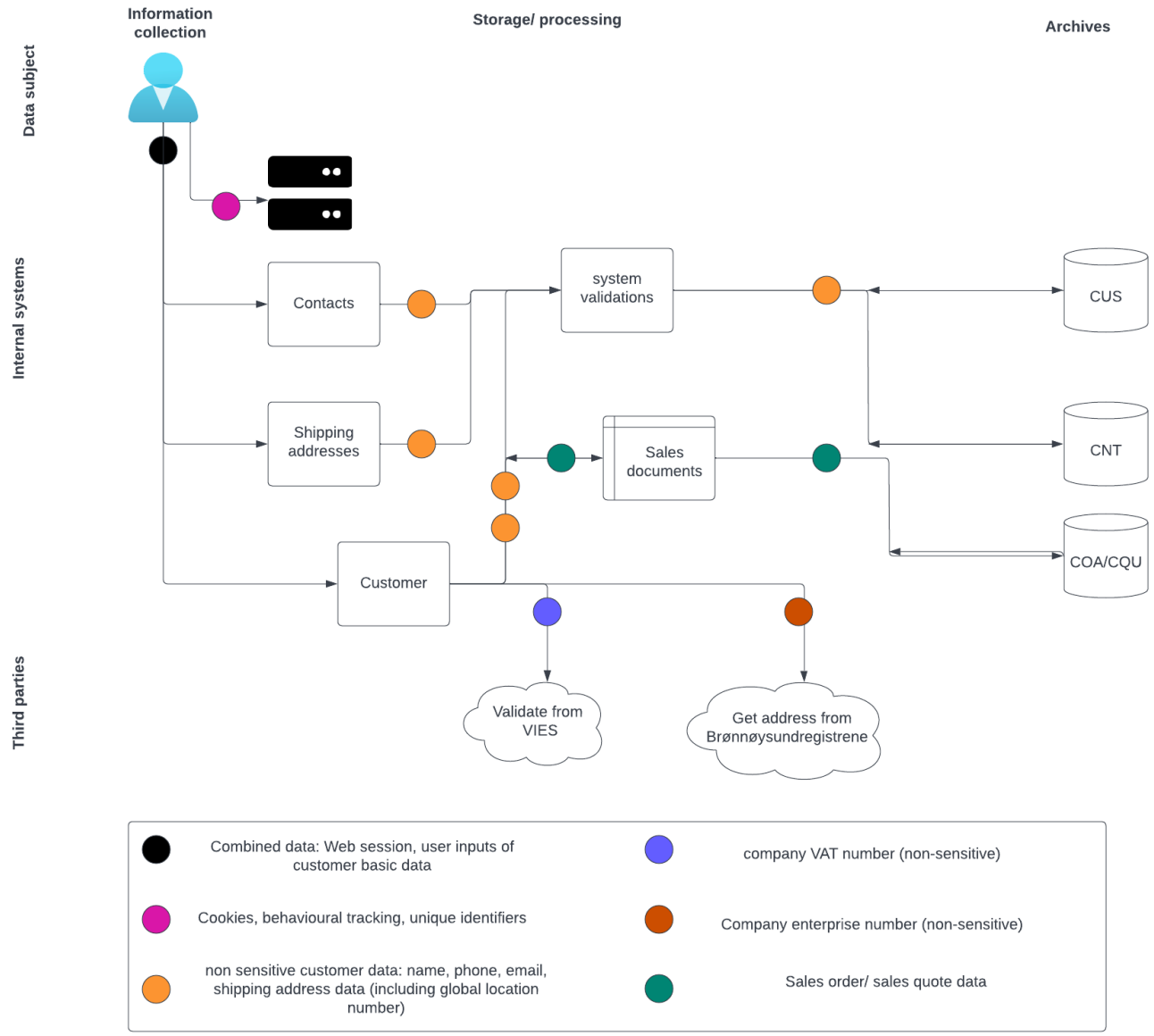
Personal data journey map

Print Cloud data flow





Rambase CRM data flow



Current risks

https://projectportal.99x.io/projects/90/risks

- lack of knowledge on information privacy
  - maturify and external docs
  - get relevant docs from https://99xtech.sharepoint.com/process/Documented Procedures/Forms/AllItems.aspx
  - add these docs to onboarding plan
- lack of knowledge on information security
  - maturify and external docs
  - get relevant docs from https://99xtech.sharepoint.com/process/Documented Procedures/Forms/AllItems.aspx
  - add these docs to onboarding plan
- lack of knowledge on software quality
  - maturify and external docs
  - get relevant docs from https://99xtech.sharepoint.com/process/Documented Procedures/Forms/AllItems.aspx
  - add these docs to onboarding plan

Security Awareness

GDPR compliance guide

GDPR compliance guide

Introduction

The General Data Protection Regulation (also known as GDPR) is a European law which was introduced on 25th of May, 2018, to ensure the protection of personal data and to verify user rights of people in Europe.

Video resource: https://www.youtube.com/watch?v=j6wwBqfSk-o (GDPR: What Is It and How Might It Affect You?)

Entities

- Data controller: Defines the purpose of data processing
- Data processor: Stores or processes personal data for a data controller/controllers
- Data subject: The owner of the personal data

Roadmap

1. Awareness - Common understanding of the law by the stakeholders and management.
2. Identify personal data - Create a data mapping about personal data
3. Review data control - In-depth review about how data are controlled by the system/ current architecture
4. Gap analysis - What is missing compared to GDPR requirements?
5. Implementation - Implement missing action items which are required for the compliance
6. Monitoring - Verifying status about compliance with the time via internal/external auditing

Checklist for websites/web apps

- Privacy Policy and Terms and Conditions pages
- Consent/ Permission
- Cookies
- Data management
- Age checks
- Data portability
- Temporary data

Further reading

- https://gdpr.eu/checklist/
- https://gdprchecklist.io/
- https://dataprivacymanager.net/difference-between-data-controller-and-data-processor/

Decision/Incidents Log

Use this table to document any and all decisions related to access PII in production systems, or any incidents that prompted anyone from the Hatteland team to access production data/systems. Clearly indicate the Decision or the Incident and dates mentioned in the table below. When this is posted in the customer confluence, get one of your POs to sign off on the acknowledgement column to indicate that they are aware of this decision/incident.

Incident/Decision	Discovered Date	Notified Date	Action Date	Action	Acknowledgement

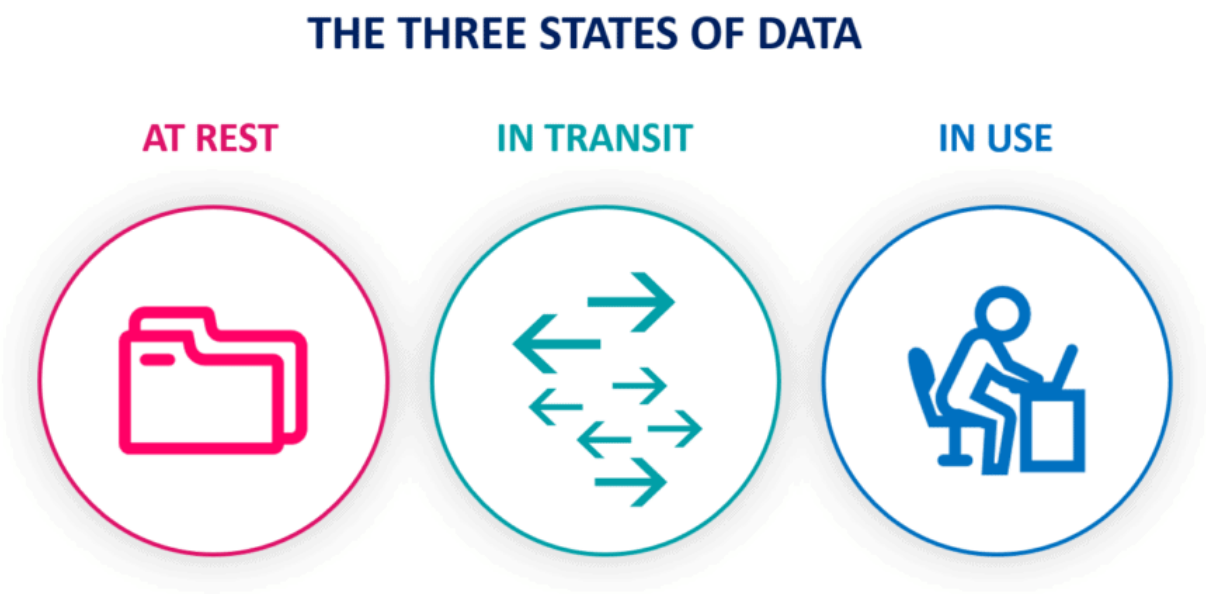
Enterprise Privacy Architecture

The term **privacy** is the right to be let alone, or freedom from interference or intrusion. Information **privacy** is the right to have some control over how your personal information is collected and used.

Privacy boundaries are not limited to users of the system . It can apply to many levels

- Users
- Organizations
- Development / Production
- Micro Service Boundaries
- Application Boundaries
- Infrastructure Boundaries (Subnets , Public-private gateways)

Rambase architecture is designed in a way that it has end to end fullstack modularity. Therefore it has special abilities to isolate boundaries mentioned above without compromising the privacy of information in 3 different states in Rambase system



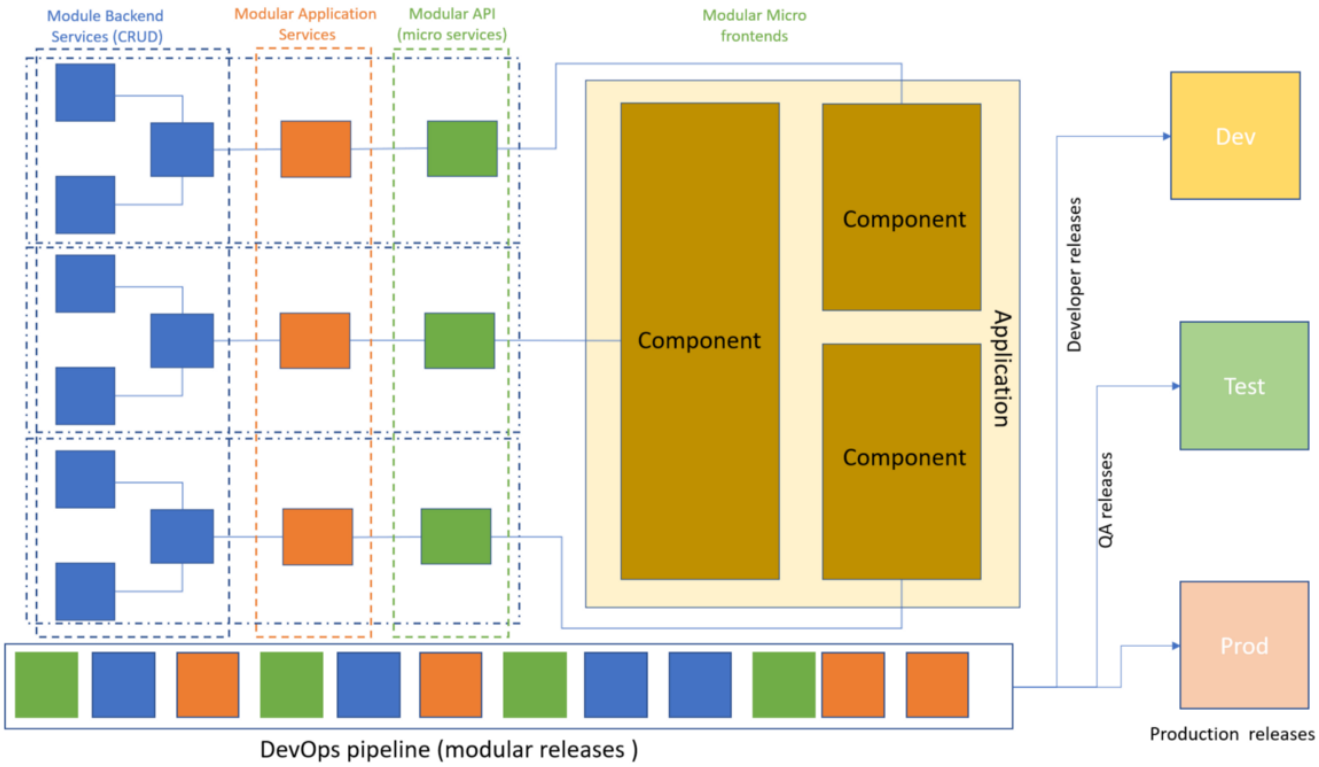
- **Data at rest**: By this term we mean data that is not being accessed and is stored on a physical or logical medium. Examples may be files stored on file servers, records in databases, documents on flash drives, hard disks etc.
- **Data in transit**: Data that travels through an email, web, collaborative work applications such as Slack or Microsoft Teams, instant messaging, or any type of private or public communication channel. It's information that is traveling from one point to another.
- **Data in use**: When it is opened by one or more applications for its treatment or and consumed or accessed by users.

Data at Rest Security	Role base acess control for Archives Field level permision Data are stored on will secured data bases Perimeter level security in private subnets that are acces trogh services behind the firewall Secured File storage with primeter level security
Data in Transit Security	Allowed only for authenticated rambase users Secured web service end points Encripted network channel (https)
Data In use	Very powerfull permission control system that can control permission at different levels • Workflow level permission control • Archive level permission • System level permission

	• Deployment stages level permission • Role base permission Accees only for authenticated users Authentication and authorization there Rambase security SDK

Levels of Isolation In the Architecture

1. **Full-stack modularity**



Being modular at all levels Rambase platform architecture decomposes your traditional enterprise from the top to bottom into smaller self-contained Apps and Components). These are designed to perform specific feature areas in ERP workflow. By wiring, this modular self-contained feature(Micro-frontends) together will formulate complex ERP workflows.

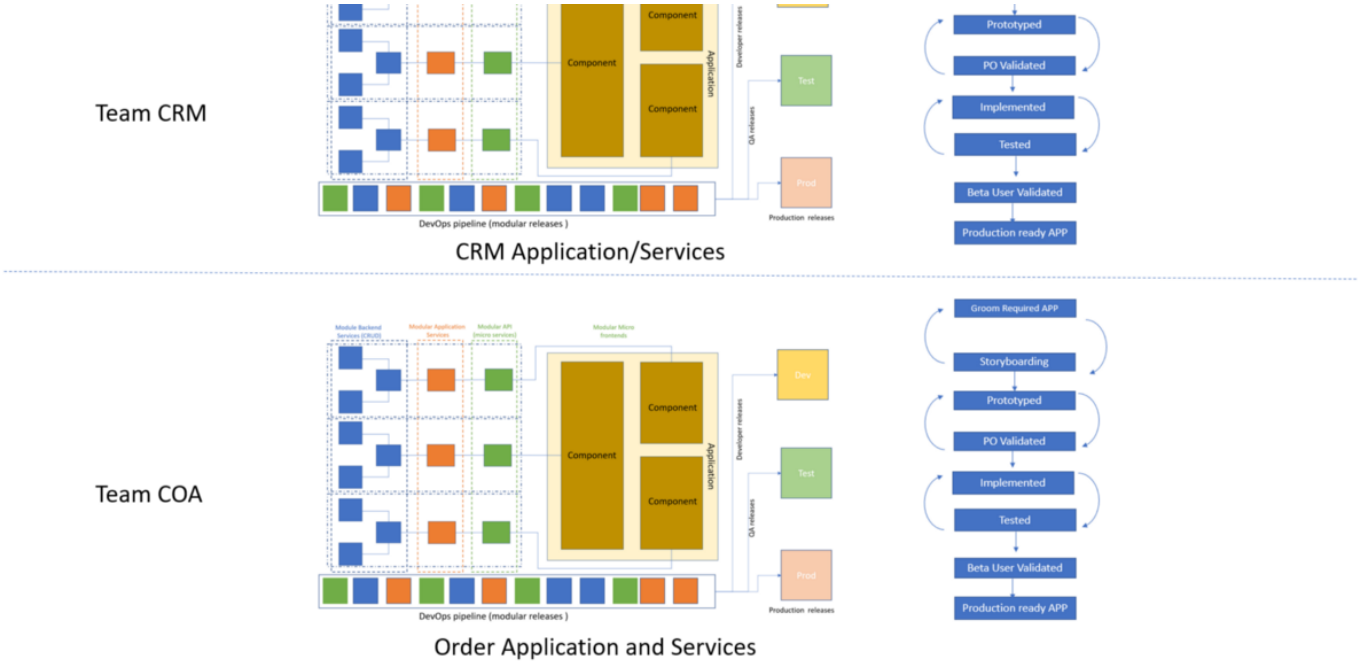
- Infrastructure Modularity (Virtualization/Containers)—In progress
- Application Server Modularity (Domain aligned business logic + CRUD)
- Service Modularity (Domain aligned Micro-Serviced services)
- Modular API Gateway (Domain aligned REST API)
- Application Modularity (Domain aligned Micro-front ends using Apps and Components)
- DevOps Modularity (Efficient DevOps pipeline which can independently deploy Application Server modules, APIs, Applications, Component …)
- User Modularity (Deployment stages for Testers, Developer, Production)
- Modularity in Security aspects (Module level Roles and Permission Assignment)
- Code-level modularity
- Making application development technology-agnostic—In progress (Support for web components specification in progress. This will enable us to develop application frontend using any of the preferred front end technology stacks which support web components)

This modularity allows isolation of runtime systems at each level. So information flow in each use case is resides withing the running microservice instances so can information can not be leaked between microservices

2. Team level Isolation

New modular Micro Frontend Architecture has allowed allocating multiple teams developing and deploying independent applications in isolation without interfering with other's release cycles. These teams are very focused and specialized in a specific micro area of the entire product who is responsible for the full-stack development of the microservice.

Teams develop applications without affecting each other release cycles



Each application development team has

- A business Product Owner who is responsible for the grooming and validating requirement of the application.
- Technical Product Owner Who is responsible for code and adhering to Architecture.
- Full Stack Development Team who is responsible for developing and testing applications as per the requirement.
- UX/UI engineer who is responsible for the User Experience of the Application.

In addition, WIP(Work In Progress) is managed in a combination of SCRUM/Kanban-style practices. Applications are groomed, developed, prototyped, testes with constant feedback of different technical and business stakeholders through demos and discussion.

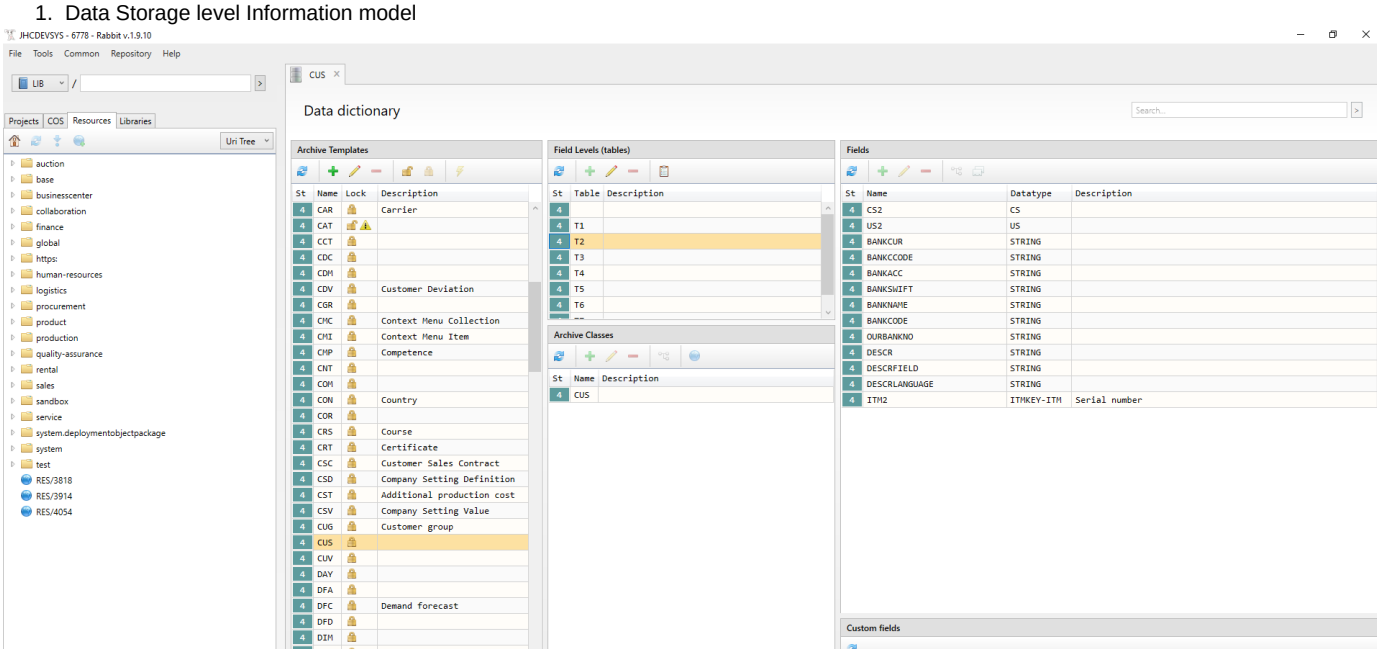
Traceability in Data

Wel defined Infomation models are maintained with end to end runtime and design time tracability

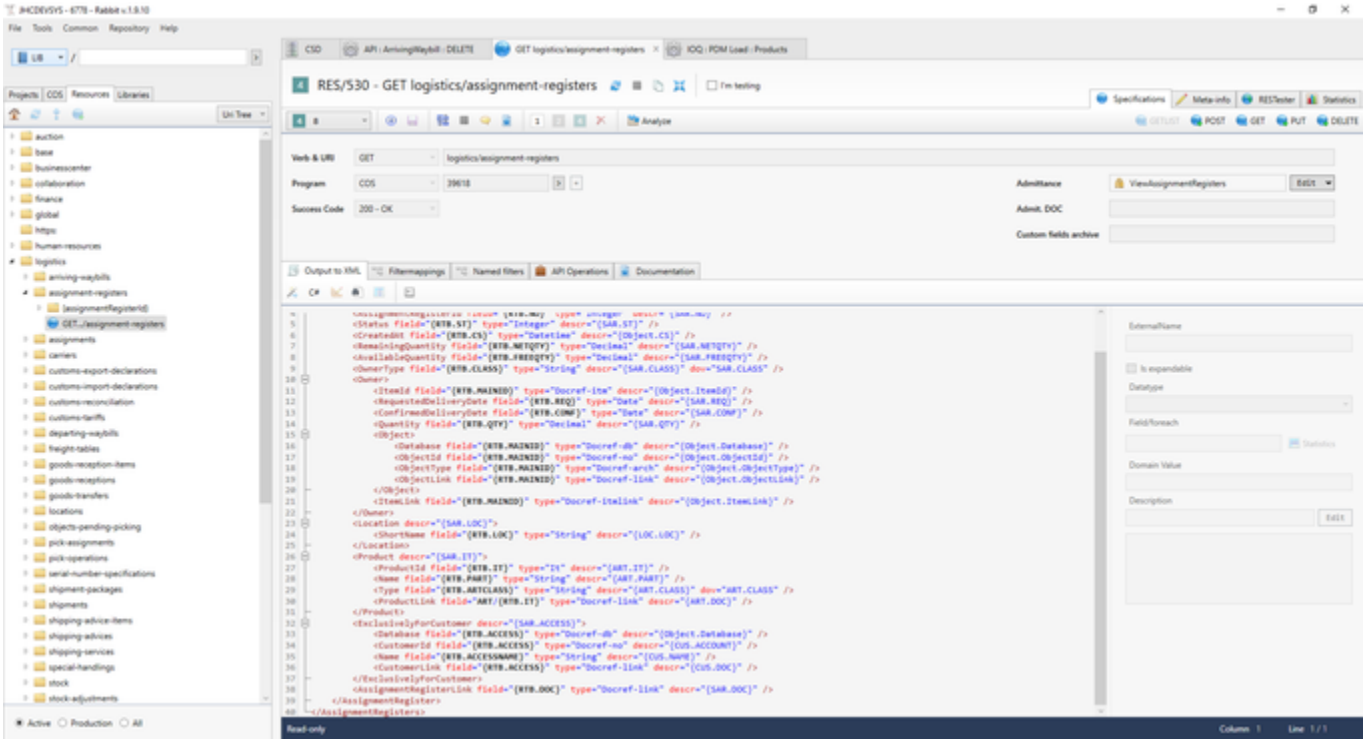
- Data Storage Level
- Middleware Level
- API Service Level
- Application Level

Design time Traceability

1. Data Storage level Information model



82. API level information models



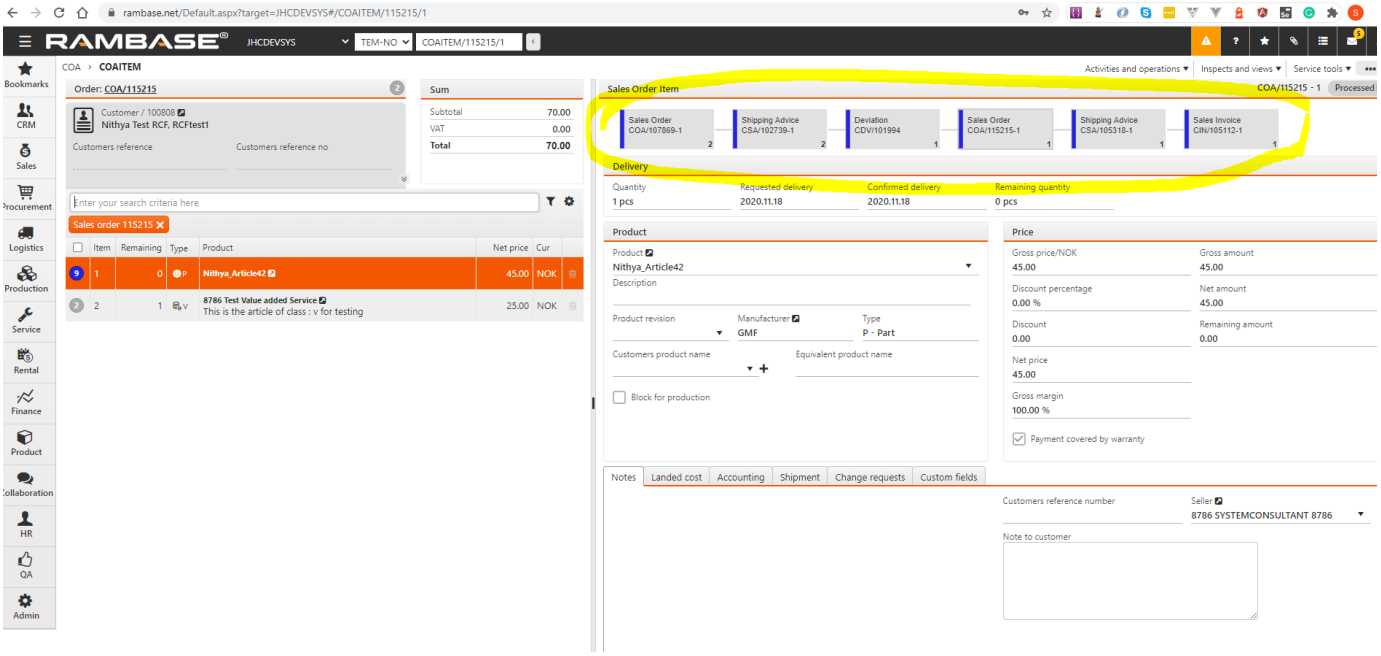
3.

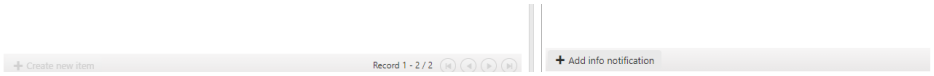
Run time Traceability

- Database traction recorded
- Execution statistics
- Trace logs

Can be used to achieve runtime traceability in data

Application Level Traceability in Data related to business transactions workflows





Achiving GDPR Compliance

https://rambase.atlassian.net/browse/RC-567

Applications	Fields	Reason	Handling	Disposal
General	PID Email Password	Authenticate the user to the system	Authenticate the user to the system with appropriate encryption.	Upon request
CONTACT	Source	Source of which the personal information was obtained	Rambase on-premise database	Upon request
CONTACT	Legal basis	The legal basis for processing contact's data	Rambase on-premise database	Upon request
CONTACT	Privacy policy accepted	Accepts privacy policy	Rambase on-premise database	Upon request
CONTACT	Consent to marketing	Accepts marketing consent	Rambase on-premise database	Upon request

PHR Policy Suggestions with GPT3 Models

PHR Suggestions Project.mp4