# Start

1. Hardware requirements - 8 GB RAM minimum, as we will be running so many services, database, message queue, webserver, caching server all locally etc..
2. Give email address to Mick to allow him to add us to JIRA/Confluence
   1. Check JIRA by <https://elevate.atlassian.net>
3. Signup to github
4. Give github account to Mick to allow him to add us to elevate gihhub account
   1. Better to not to use long paths for a folder with elevate project, as some problems are possible. Better to create the folder on some disk’s root.
   2. Mick recommends sticking to the command line
   3. Vlad and Oleg also use TortoiseSVN
5. Install Visual Studio 2012
   1. Better to make it to be always opened as Administrator
6. Install ReSharper
7. Install beta.join.me to have online meetings with screen sharing
8. Install mySql
   1. Also install Workbench
9. Install RabbitMQ
   1. It requires to install Erlang first
10. Install MongoDB
    1. Also install MongoVUE - is good for visual management
11. Install Ruby
12. Install hipchat – (elevate internal chat system)
    1. Mick sends email invite to it
13. Install Java
14. Go to root elevate folder. Type ‘rake –T’
    1. If error message, then:
       1. Go to folder ‘tasks’, run file install-gems.bat
       2. Back to root folder, run command ‘gem install rake’
       3. **There is also a documentation** [**https://elevate.atlassian.net/wiki/display/ELEVATE/Local+Development+Environment+Setup**](https://elevate.atlassian.net/wiki/display/ELEVATE/Local+Development+Environment+Setup)
       4. If rake –T still doesn’t work, run it manually by typing in project’s root:
          1. gem install log4r
          2. gem install rubyzip
          3. gem install albacore
          4. gem install right\_aws
          5. To run it quicker, we can use parameters ‘ --no-rdoc --no-ri’
          6. Then run ‘rake –T’ again
15. Deploy database
    1. Go to project root folder, then scripts\sql\create-schema-and-user.sql
       1. Run this script in mysql – this will create database and user
    2. Then go to project root and run: rake db:migrate – this will fill database with data
       1. If doesn’t work – we can run it manually by ‘project root\output\app\console\Elevate.Console.exe migratedatabase’
       2. It was also a problem with primary keys, so Mick gave us a big sql-script file to create the db. So, need to:
          1. Drop elevate database from mysql firstWe
          2. Run the script from mysql console by command ‘source d:\08-11-2012-Prod-Cleaned-Passwords-Reset.sql’. (the script file is in the same folder this instruction is in)
          3. After that we ran rake db:migrate.
16. in the root call start\_vendor\_services.bat. Give it 20 seconds and then navigate to <http://localhost:8000/elevate>. This starts up apache solr & memcached
17. For rabbitMQ – go to localhost:15672. Default login is guest guiest
    1. Also need to activate the management plugin by:
       1. Open RabbitMQ console
       2. Type ‘rabbitmq-plugins enable rabbitmq\_management’
       3. Stop and run the RabbitMQ service
    2. Need to create virtual host
       1. In RabbitMQ go to Admin
       2. On the right – ‘Virtual Hosts’
       3. Add new virtual host ‘elevate’. It will appear in the hosts list
       4. Click elevate in the list
       5. Click ‘set permissions’
18. Go to elevate root and run symlinks.bat
19. Then run the web application by

(if problems with finding files in temp folder – give the rights for this folder to the account running the iis – most probably - \IIS\_IUSRS)

* 1. Go to http://localhost/employers/login
     1. user: md@elevatedirect.com
     2. pwd: $Password123
  2. Go to: http://localhost/contractors/login
     1. to get a username for contractors: query: SELECT EmailAddress FROM elevate.contractors\_contractor;
  3. Go to <http://localhost/admin/login>
     1. username = email
     2. password: $Password123
  4. Navigate around the application...
  5. At the root run: start\_services.bat
     1. that runs all the background services.

# Test tasks

1. Here's a sample Mick put together a while back <https://github.com/mickdelaney/masstransit-rabbitmq-sample>
   1. fork it on github into your own accounts, then get the latest and you can use it to get started, make changes and store it etc (back on github). It will help you to understand it,

elevate's code is essentially the same but might be more difficult to understand initally...

1. Task 1 - modify the sample
   1. add new messages,
   2. add a website, and have it publish & subscribne to messages..
   3. turn off the subscriber, but have the pubishing app still send messages..
   4. these messages will be stored in the broker (rabbitmq), open up the ui and look in the subscribers queue..
   5. you'll see the messages in there waiting
   6. then start the subscriber, it will then receive the pending messages in the queue...
2. Task 2
   1. add a saga to the sample - have something like this
   2. website 1 has a form to create a customer
   3. the service creates the customer, (keep the data in memory for now)
   4. then the service fires a message to WEbsite 2 (customers i think we called it).
   5. this website then needs to send another 'Authorize Customer' message
   6. once the service receives the Authorize message, it then publishes a 'Customer Account Created' message..
3. Task 3
   1. Take each app in the solution
   2. we want each app to create a WindsorContainer on startup, load it full of components and use them during the life time of the app..
   3. so first things first, go and get the Castle.Windsor nuget package.
   4. also get the Masstransit Windsor package too, its a masstransit package with some extensions methods and utilities for working with WindsorContainer
   5. once you have them installed in the solution go ahead and create an instance of the container somewhere in each app that will be in scope until the app shuts down.
   6. <http://docs.castleproject.org/Default.aspx?Page=Installers&NS=Windsor&AspxAutoDetectCookieSupport=1>
   7. <http://docs.castleproject.org/Windsor.Registering-components-by-conventions.ashx>
   8. we use conventions in elevate so you need to use them in the the sample
4. Task 4
   1. take full user details, e.g. FirstName, LastName, EmailAddress, Age, Location  
      create a web form for it.
   2. the service should take the message and store it in mongodb.
   3. then create a selenium webdriver automated test to test the web form you created.. <http://nuget.org/packages/Selenium.WebDriver>

use the ghostdriver implementation.  
you'll need to download PhantomJS Ghostdriver  
<http://phantomjs.org/> <https://github.com/detro/ghostdriver>

* 1. detro/ghostdriver  
     Ghost Driver is an implementation of the Remote WebDriver Wire protocol, using PhantomJS as back-end

1. Task 5
   1. what we want to do is start to create a framework to test elevate with...
   2. and this is acceptance testing (browser automation)
   3. the toolset to use is, specflow, webdriver (ghostdriver + phantomjs),
   4. the key thing about this is to get the page object pattern working effectlively..
   5. reading:
      1. <http://www.marcusoft.net/2012/05/specflow-page-objects-and.html>
      2. <http://watirmelon.com/2011/03/15/specdriver/>
   6. this is the important thing.
      1. Our requirements are written in the format used by specflow
      2. when testing our website, change will happen, links will change etc, so we use the page object pattern to manage that change and reuse pages
      3. 3) ghostdriver & phantomjs makes it faster and easier to test than using a 'real' webbrowser, which is slow and difficult to automate
   7. you need to make sure you use the "page object" pattern..
   8. that's key..
   9. also make sure you are using ghostdriver which is part of the latest webdriver nuget package
2. Task 6
   1. automate the 'sign up' process in elevate..
   2. its a v important form that has to be tested every release, so would be great to get it automated...
   3. what it means is that you'll need to navigate screens..
   4. the issue is, its ajax based...i.e. client side..
   5. so using the browser path/url probably wont work
   6. its essentially 1 page
   7. so that's going to be faster..
   8. is to build a similar one in your test app, and once you’r comfortable, and we've reviewed it etc, we'll go ahead and develop it again in the elevate app..
   9. ok, basically its using historyjs - https://github.com/balupton/History.js/
   10. if you look at the demo http://balupton.github.com/history.js/demo/
   11. its using a knockoutjs template binding -http://knockoutjs.com/documentation/template-binding.html
   12. the code is in the elevate codebase...
   13. if you go to Elevate.Web.Contractors -> Client/SignUpSteps
   14. and under Views/SignUpSteps/Steps.cshtml
   15. you'll see the code..
   16. the 'template' that's bound is selected based on this code:
   17. <div class="span9" data-bind="template: { data: step, name: stepTemplate.bind($data) }">
   18. Loading...
   19. </div>
   20. if you look in PageView.js
   21. you'll see the function 'stepTemplate'
   22. stepTemplate: function () {
   23. return this.step().templateId;
   24. },
   25. that returns the current steps 'templateId'
   26. as you can see in the code above..
   27. so that gives you as string which represents the name of the template to render..
   28. each step is loaded in the PageView.js in the createSteps function
   29. createSteps: function () {
   30. return [
   31. new CreateAccountView(this, this.pageData.invitationCode, this.pageData.contractorName),
   32. new SkillsView(this),
   33. new WorkExperiencesView(this),
   34. new RequirementsView(this),
   35. new FinishView(this)
   36. ];
   37. },
   38. each element in the array is a 'StepView'
   39. i.e. it extends StepView
   40. e.g var CreateAccountView = StepView.extend({
   41. in the constructor (init function)
   42. it calls the super (base class)
   43. this.\_super(page, "CreateAccount");
   44. passing the 'templateId'
   45. this is just a name for the piece of html to render that page.
   46. so we add the steps.
   47. the knockoutJS template binding uses the 'templateId' from the current step.
   48. so when we change the current step, the template will change..
   49. probably best is to create a simple example.
   50. 3 pages.
   51. you'll need 3 templates.

# Info to learn

1. Mick has a book & videos on using git
2. <http://www.pluralsight.com>. There's 2 courses to look at:
   1. <http://www.pluralsight.com/training/Courses/TableOfContents/selenium>
   2. <http://www.pluralsight.com/training/Courses/TableOfContents/continuous-integration>
3. Knockout tutorial – it’s good
4. MassTransit
   1. <http://docs.masstransit-project.com/en/latest/>
   2. Start here <http://docs.masstransit-project.com/en/latest/overview/index.html>
   3. Also
      1. <http://www.dovetailsoftware.com/blogs/kmiller/archive/2009/08/26/producing-and-consuming-messages-using-masstransit-and-structuremap>
      2. <http://architects.dzone.com/articles/quick-start-mass-transit-and>
      3. <http://lostechies.com/chrispatterson/2009/10/29/building-a-service-gateway-using-masstransit-part-1/>
      4. <http://lostechies.com/chrispatterson/2009/10/29/building-a-service-gateway-using-masstransit-part-2/>
      5. <http://lostechies.com/chrispatterson/2009/11/01/building-a-service-gateway-using-masstransit-part-3/>
   4. Sagas
      1. <http://lostechies.com/chrispatterson/2008/08/29/managing-long-lived-transactions-with-masstransit-saga/>
      2. <http://docs.masstransit-project.com/en/latest/overview/saga.html>

# Info from Mick about project

1. IDispatch interface , IDispatchCommands is injected by our DI. We dispatch command, have results. If result.succeds – do something, if fails, do smth
   1. CommandHandler, it’s also wired up by DI. Interface IProcessCommand
   2. When the website first starts up, some code looks for every class that implements the IProcessCommand
   3. The most of the project is done with these commands and handlers
   4. Example – when we sign in, we take everything – email, password… put into command, send the command to the commandhandler. Handler looks for this person (uses repository). Also and authentication service is injected into the handler by dependency injection. We publish events
2. We use Fluent Migrator for database migration (instead of having database scripts – we use c# code) <https://github.com/schambers/fluentmigrator/wiki>
   1. In Visual Studio – open Elevate.Task\Data\Migrations – all the migrations are here
3. If you login as a contractor, an event is published (using masstransit/rabbitmq)
   1. so you need to have start\_vendor\_services.bat
   2. then start\_services.bat
   3. give it 30 seconds...
   4. then go to visual studio and debug the process Elevate.Services.exe
   5. put a breakpoint in ContractorLoggedInEventHandler Line 25: \_logger.InfoFormat("Contractor With EmailAddress {0} Has Logged In", message.Contractor.EmailAddress);
   6. this should be invoked once the login is successful
4. in elevate we're using nhibernate
   1. masstransit has an nhibernate saga persister built in too..
   2. if you look at ContractorSignUpSagaMap - that's a Fluent NHibernate Map for the ContractorSignUpSaga in Elevate
5. message subscription is based on the message type being subscribed by the app/service that wants it...
   1. if you look in elevate code... at the classs ServicesMessageBusInstaller !!!

/// <summary>

/// Scan The Service Host Assembly For All Consumes<T>.All

/// Register Them In the Container

/// </summary>

void RegisterMessageHandlersInContainer()

{

var consumerType = typeof (Consumes<>.All);

\_container.Register

(

consumerType.ComponentsOfTypeInAssembly(\_serviceAssembly).Configure(c => c.LifeStyle.Scoped()),

Component.For(typeof (ISagaRepository<>)).ImplementedBy(typeof (NHibernateSagaRepository<>))

);

}

* 1. here we scan the assembly for any class that implements Consumes<>.All
  2. we then register those classes in the container..
  3. later masstransit scans the container for all those classes
  4. at startup, it scans the container, for each class it finds it then knows that the servivce wants to subscribe to the message
  5. so as long as the scanning code is correct, all the developer needs to do is add another class that implements the interface Consumes<>.All
  6. if you look in ElevateServiceBusFactory Line 30 - you see that MT (masstransit) is being told to scan the IoC container for consumers... sbc.Subscribe(subs => subs.CreateMasstransitConsumersFromContainer(\_container));
  7. conceptually its like this:
  8. You have an assembly (.net). It contains classes like: public class CustomerMessageSubscriber : Consumes<CustomerMessage>.All {}
  9. you may have 10 of those for example: at startup: you scan the assembly using reflection: it returns the 10 classses, you then configure your dependency injection container (also called an IoC container) by adding those classes into the container
  10. now you container contains the 10 classes.
  11. you then tell MT to look in the container for any classes implementing Consumes<>.All
  12. it finds the 10 classes.
  13. then it has 10 messages it needs to subscribe to in rabbitmq
  14. the alternatiive is that you would have to manually subscribe every message, which is error prone, awkard and a waste of time really.

1. About Windsoe container
   1. if you look in ElevateHttpApplication<T>
   2. for web apps, the container is stored like this:

public static IWindsorContainer Container { get; private set; }

IWindsorContainer IContainerAccessor.Container

{

get { return Container; }

}

* 1. if you look into ServiceHostBase
  2. protected IWindsorContainer Container;
  3. essentially is a global object.. but we dont access it that way.
  4. once its configured, we then make it available via the container itself, in addition to it being a global
  5. <http://docs.castleproject.org/Default.aspx?Page=Installers&NS=Windsor&AspxAutoDetectCookieSupport=1>
  6. <http://docs.castleproject.org/Windsor.Registering-components-by-conventions.ashx>