Form Builder MVP

Contents

[Landing Page 3](#_Toc494620037)

[MY FORMS 3](#_Toc494620038)

[“MY FORMS” menu option is provided on the landing page. 3](#_Toc494620039)

[“MY FORMS” Page 3](#_Toc494620040)

[Form Actions 3](#_Toc494620041)

[Create Form 4](#_Toc494620042)

[Form Builder 5](#_Toc494620043)

[Form Elements 5](#_Toc494620044)

[Adding Controls 7](#_Toc494620045)

[Control Properties 7](#_Toc494620046)

[Saving Form Design 9](#_Toc494620047)

[Publish 9](#_Toc494620048)

[Form Submission 9](#_Toc494620049)

[Viewing Form Submissions 9](#_Toc494620050)

[Submissions UI 10](#_Toc494620051)

[Planning 11](#_Toc494620052)

[Marketable Features 11](#_Toc494620053)

[Database Design 12](#_Toc494620054)

[Collections 13](#_Toc494620055)

[Database Initalization 13](#_Toc494620056)

[Security 13](#_Toc494620057)

[Setting up Development Environment 13](#_Toc494620058)

[Install Software 13](#_Toc494620059)

[UI Design 14](#_Toc494620060)

[Approach 14](#_Toc494620061)

[Setting up Angular Material 14](#_Toc494620062)

[REST APIs 15](#_Toc494620063)

[Guest Authentication 16](#_Toc494620064)

[16](#_Toc494620065)

[16](#_Toc494620066)

[16](#_Toc494620067)

[16](#_Toc494620068)

[New 16](#_Toc494620069)

[Existing 17](#_Toc494620070)

[Angular Mysteries 17](#_Toc494620071)

[Custom Icon Font 17](#_Toc494620072)

[Deployment 18](#_Toc494620073)

[Building FormDesigner 18](#_Toc494620074)

[Auto Build 18](#_Toc494620075)

[Manual Build 18](#_Toc494620076)

[Serving FormDesigner from Node 19](#_Toc494620077)

[Deploying MongoDB using MongoDB Atlas 19](#_Toc494620078)

[Deploying Node App Server 19](#_Toc494620079)

[Using AWS Elastic Beanstalk 19](#_Toc494620080)

[Using AWS EC2 20](#_Toc494620081)

[Using Cloud Foundry 21](#_Toc494620082)

[Getting Started 21](#_Toc494620083)

[Changes made to deploy to Cloud Foundry 21](#_Toc494620084)

[Notes 22](#_Toc494620085)

[Performance Improvements 23](#_Toc494620086)

[Form Builder Server 23](#_Toc494620087)

[Initial Perf Test 23](#_Toc494620088)

[Nodejs Memory Model 24](#_Toc494620089)

[Angular Performance 24](#_Toc494620090)

[Takeaway: 25](#_Toc494620091)

[Form Designer Client 26](#_Toc494620092)

[General 26](#_Toc494620093)

[Workflow – User 26](#_Toc494620094)

[Workflow - Design 26](#_Toc494620095)

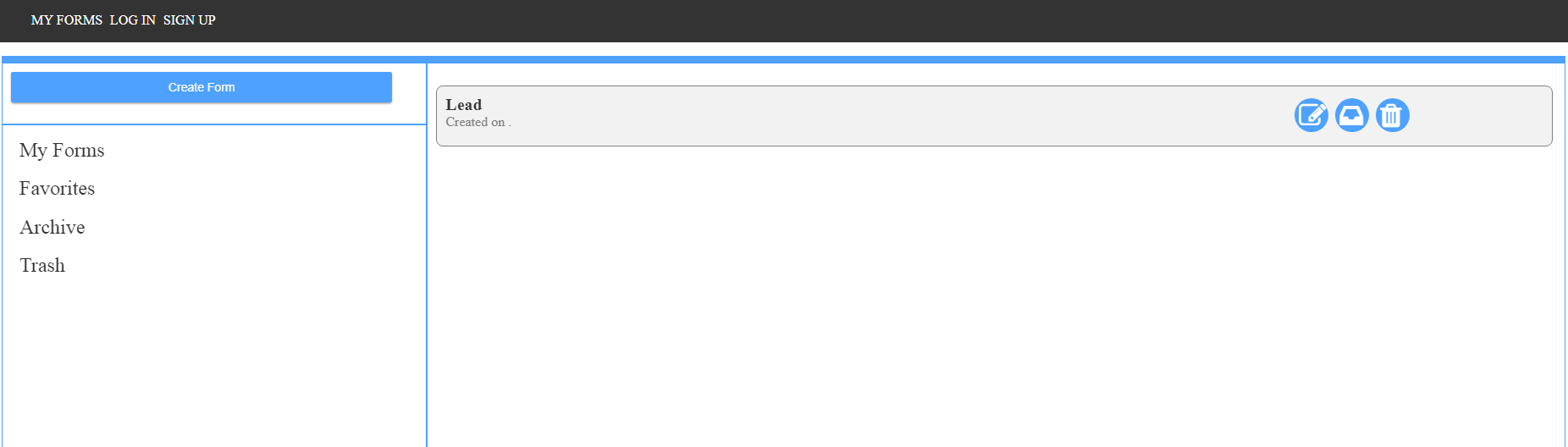
# MY FORMS

## “MY FORMS” menu option is provided on the landing page.

## “MY FORMS” Page

User doesn’t need to sign up or login to access “MY FORMS” page.

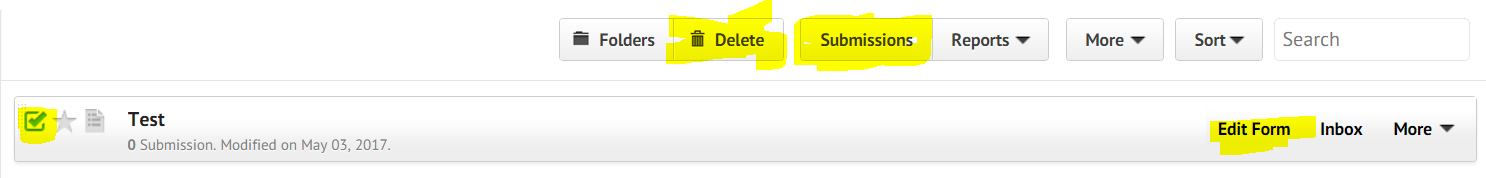
NOTE: We need a way of identifying the guest user so that the forms can be presented to the user when he/she visits “My Forms” page. One way of doing this is to use session cookie.



1. Show list of created forms.
2. Create Form button at the top left.

### Form Actions

When user selects a form, action menu is shown :



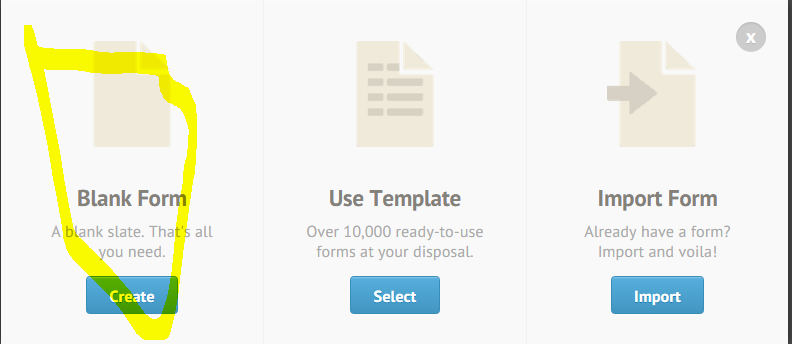
1. Allow user to select a form
2. Delete form
3. Check submissions
4. Edit Form option on the right

# Create Form

User needs to click on “Create Form” button.

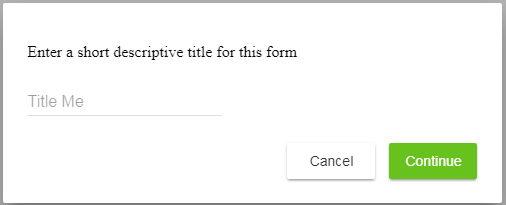


1. Form Template Selection dialog is shown. For MVP, we will do only “Blank Form”.



1. User clicks on “Create” button.

“Form Name” dialog is shown asking user to enter form name

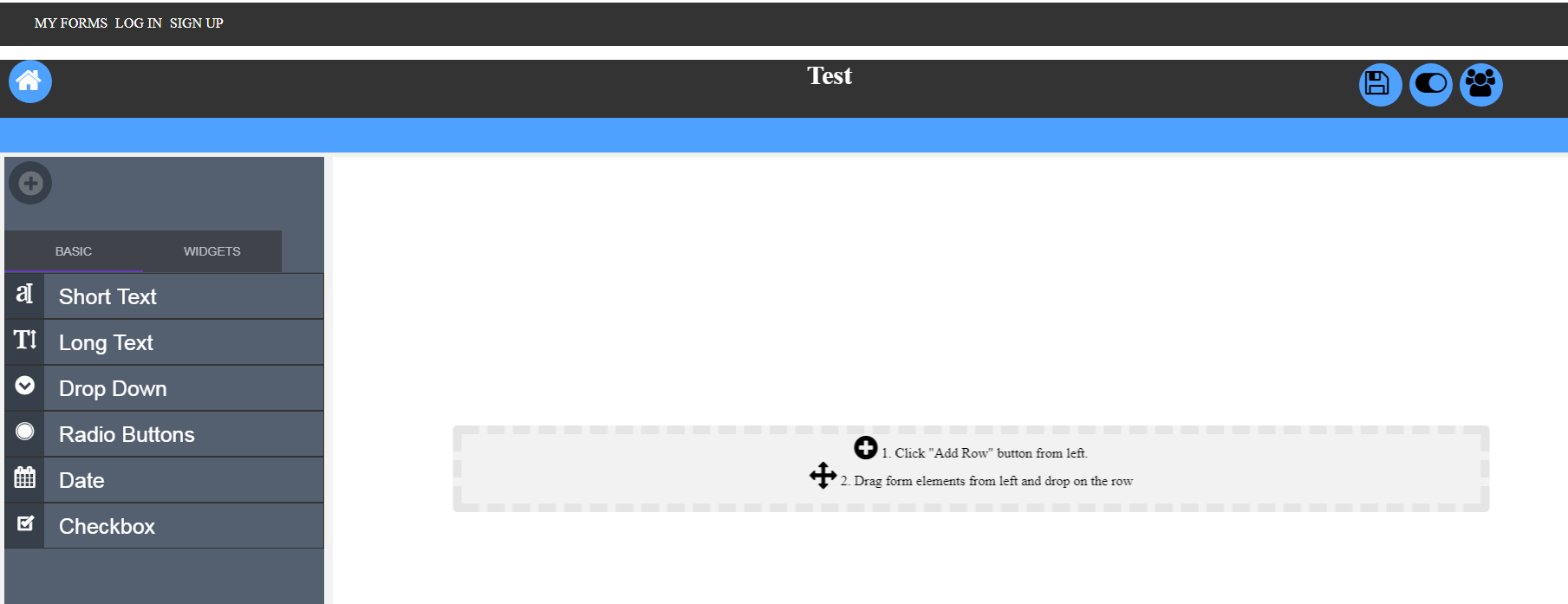


Clicking on “Cancel” closes the popup.

Clicking on “Continue” takes the user to the Form Builder UI.

“Continue” button is disabled if there is no text entered in “Title” textbox. When user enter any text, it gets enabled.

# Form Builder



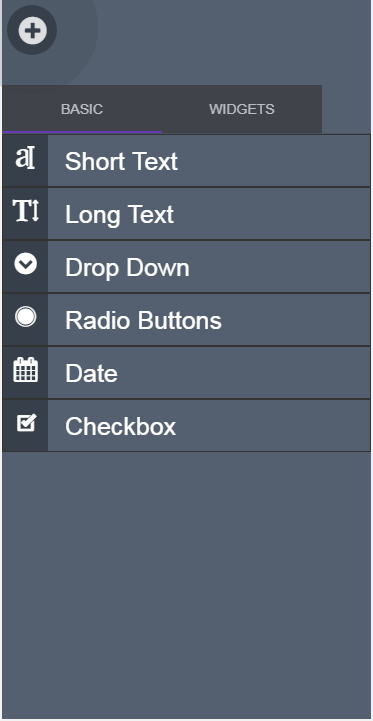
Above is the initial blank canvas for Form Builder.

MVP Features:

1. We will maintain the above layout i.e Header, Menu Bar, Main Canvas Page
2. Header:
   1. Logo on the left
   2. Form Title
3. Menu Bar
   1. Build
   2. Publish
   3. Preview Form
4. Main Canvas Page
   1. Add Form Element Panel. We will not show the “+” button instead show the controls panel directly on the left.
   2. No “Add Your Logo”, “Add New Page Here”
   3. No “Paint” icon as shown above on the right side.

## Form Elements

Form Elements (Controls Panel) will be shown on the left side.

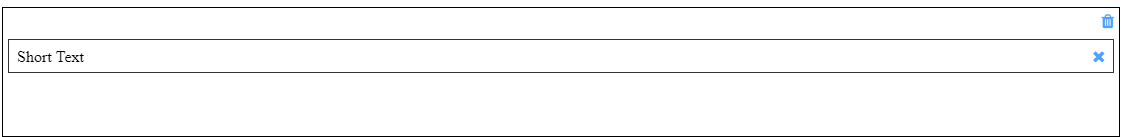


MVP Features:

1. Only Basic tab
2. No ‘X’ button
3. Only Quick Elements
   1. Short Text
   2. Long Text
   3. Dropdown
   4. Checkbox
   5. Radio
   6. Text
   7. Date
   8. Number

# Adding Controls

User has to drag and drop a control onto the center content. We will continue to have “Add Row” option.

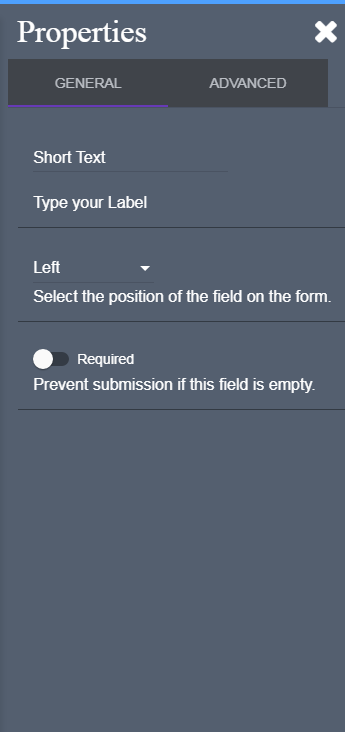


Show 2 options as shown above:

1. Properties
2. Delete

# Control Properties

Control properties UI will be shown on the right side.



MVP Features:

1. General and Advanced tabs
2. General
   1. Label
   2. Required
   3. Placeholder text
   4. Read Only
3. Advanced
   1. Control specific properties

# Saving Form Design

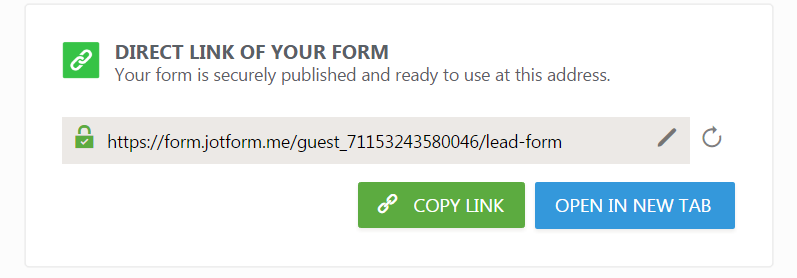
We will provide a “Save” button which will save the form. No auto saving option for MVP.

# Publish

1. Publish option on the menu bar.



1. Clicking on “Publish” shows the following UI:



# Form Submission

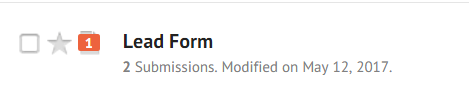
User needs to open the published form url in a browser. The form will be shown to the user. User can enter the details and click “Submit”.

On “Submit” any validations will be shown

On successful validations, form data will be saved and a message will be shown to the user.

# Viewing Form Submissions

The submitted form data by the users will be shown on the “MY FORMS” page.



1. Number of submisisons will be shown to the user as shown above
2. User selects the form and clicks on “Submission” button to view the Submissions



## Submissions UI

1. Maintain the same layout
2. Only Delete option for MVP
3. Form Submissions will be shown in LIFO order i.e latest submissions first and so on
4. User can use the navigation arrows to switch between different submissions

# Planning

## Marketable Features

1. ~~Design and Architecture~~ (6) **DONE**
   1. Tech stack
      1. FrontEnd – Angular , TypeScript, JavaScript, Angular Material, SCSS, Angular CLI, Jasmine, NPM, Chrome Dev Tools, VS Code,
      2. Server - Visual Studio, Node JS (> 7.6.0), Express, ES6
      3. Database – MongoDB
      4. Source Control – git , GitHub, code structure
      5. Project Management – Visual Studio Online
      6. Cloud – Azure
      7. CI/CD – Chef, Puppet, Travis
   2. High level design
      1. Front End
         1. Component and module architecture
         2. SCSS framework - TEJAS
         3. ~~UI wrapper~~ - **DONE**
         4. ~~Routing -~~ **DONE**
            1. Primary Routes (<router-outlet> in app)

home

Myforms

Design/:id/edit (design/0/edit for new)

Submissions/:id

Preview/:id

* + - 1. ~~Setup base project -~~ **~~DONE~~**
         1. FrontEnd

FormBuilder

App

Myforms

Formdesigner

Submissions

Preview

Shared

FormRenderer

UI

Router

FormSubmitter

* + 1. Server
       1. ~~Design RESTful API and document~~ - **DONE**
       2. ~~Authentication using cookie/header~~ - **DONE**
       3. ~~Setup base project~~ - **DONE**
    2. ~~Database -~~ **~~DONE~~**
       1. ~~Data modelling~~
       2. ~~Indexing~~
       3. Security
  1. ~~Setup projects~~ **DONE**
  2. ~~Source code control~~ **DONE**
  3. Identify tech spikes
     1. ~~Using Angular CLI. Setting up. –~~ **~~DONE~~**
     2. ~~How to not set up git when creating using cli? Not possible. Instead delete .git folder. - DONE~~
     3. ~~Production build of Angular using CLI. How to deploy angular prod build to server. - DONE~~
     4. using Node for publishing angular prod build
     5. ~~Deploying Node to Azure – Do it as part of Deployment MF~~
     6. ~~CI/CD using Azure and GitHub - Do it as part of Deployment MF~~
     7. ~~Designing UI wrapper module so that underlying UX framework can be easily swapped. For eg: Angular Materia, Kendo UI~~

1. Landing Page (10)
2. My Forms (20)
3. Form Builder (New, Edit) (40)
4. Save Form (10)
5. Publish Form (20)
6. View Form Submissions (40)
7. Do Form submissions (40)
8. Deployment (40)
   1. <https://www.visualstudio.com/en-us/docs/build/get-started/aspnet-4-ci-cd-azure-automatic>
   2. MongoDb security
   3. Understand domain registration and pricing
   4. Resetting base path for Angular apps. Its set by default to <base href=”/”> in index.html

Using Angular CLI: ng build –base-href /formbuilder/

## Database Design

We will be using MongoDb as the database.

### Collections

|  |  |  |
| --- | --- | --- |
| Name | Schema | Index |
| formsMeta | {  Id:’’,  userId:’’,  formName:’’,  meta:{}  } | Id, userId |
| formsData | {  formId,  data:{  }} | formId |
| Users | Id  isGuest  loginId  salt  hash | Id |

### Database Initalization

We need to write code when creating MongoDB to set up the following:

1. Create above indexes when creating database and collections. Use db.collection.createIndex API

### Security

TBD in deployment

# Setting up Development Environment

## Install Software

|  |  |  |
| --- | --- | --- |
| Node | >7.6.0 | Nodejs.org |
| NPM |  |  |
| MongoDB |  |  |
| VS Code |  |  |
| Visual Studio 2015 > |  |  |
| Node JS Tools for Visual Studio |  |  |
| Git |  |  |
|  |  |  |

**## Scaffolding for FormBuilder and FormSubmitter**

1. Requirements - Node, NPM, Angular CLI
2. To generate app:

**ng new FormBuilder --prefix fb --routing -sg true --style scss**

|  |  |
| --- | --- |
| **--style scss** | **Styles should use SASS** |
| **--prefix fb** | **Use fb prefix** |
| **-sg true** | **Skip git** |
| **--routing** | **Add a default routing module** |
|  |  |

1. To generate artifacts
2. To serve

**ng serve -o –w**

1. To lint

ng lint –-format stylish

1. To build

Ng build

**## VS Code Settings**

File > Preferences menu or the Command Palette (Ctrl+Shift+P) search for File Icon Theme

Select Seti

# UI Design

## Approach

Create UI module which provides wrappers for underlying UI elements. This allows swapping the UI framework without impacting the entire application.

UI Library

(Kendo, Ng Material)

UI Module

App/Module

## Setting up Angular Material

1. Install 2 packages : angular material, angular animation

Npm install –save @angular/material

Npm install –save @angular/animations

1. Make changes to module to import angular material modules

// Angular Material modules

import {MaterialModule} from '@angular/material';

import {BrowserAnimationsModule} from '@angular/platform-browser/animations';

@NgModule({

imports: [

CommonModule,

MaterialModule,

BrowserAnimationsModule

]

1. Add the following in styles.scss

@import url('https://fonts.googleapis.com/icon?family=Material+Icons');

@import '~@angular/material/prebuilt-themes/deeppurple-amber.css';

<https://github.com/angular/angular-cli/wiki/stories-include-angular-material>

# REST APIs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Operation | URL and Method | Headers | Request | Response |
| Get forms for a user/guest | GET  /api/forms/{includeMeta}  includeMeta – 1/0  If 1, send entire meta  If 0 , send formid, form name | **Request**:  Content-type : application/json  Accept : application/json  X-Auth-Header : username:<username>  **Response**:  X-Auth-Header : username:<username>  Cache-Control: private; max-age=3600 |  | **Success:**  200 OK  No username:  Return []  With username:  IForm[]  **Failure:**  400/500  Error code  Error message  In case of no user name, generate a random unique username and send in response header:  X-Auth-Header |
| Get form for a user/guest | GET  /api/form/:id | **Request**:  Content-type : application/json  Accept : application/json  X-Auth-Header : username:<username>  **Response**:  Cache-Control:  private; max-age=3600 |  | **Success:**  200 OK  No username:  Return {}  With username:  IForm  **Failure:**  400/500  Error code  Error message |
| Save form meta | POST  /api/form/:id | **Request**:  Content-type : application/json  Accept : application/json  X-Auth-Header : username:<username>  **Response**: | Form meta JSON | **Success**:  201 Created  200 OK  Return Form meta  **Failure:**  400/500  Error code  Error message |
| Save form data |  |  |  |  |
| Get form data |  |  |  |  |
| List submissions for a form |  |  |  |  |

## 

# Identity Management

Form Builder implements following types of authentication:

1. Guest Authentication
2. Token-based Authentication
3. External Provider Authentication (Facebook)

## Guest Authentication

In Guest Authentication, user can use the app features without the need to create user account. It is meant to provide an easy way for users to try the app.

Server

**Client**

Auth Middleware

API Routes

### 

Users

Local Storage/Cookie

### New

1. User clicks "My Forms"
2. Client reads local storage/cookie for guest id
3. Calls server API:/ api/forms. Passes the guest id (cookie/header)
4. Request reaches server.
5. Auth middleware receives the request (Auth middleware is configured before all API routes)
   1. Reads the userid (request.cookie or request.headers)
   2. If userid is blank, implies new user. So generate a random unique userid
   3. Save the new user in users collection in MongoDb
   4. Sets request.user = {user:<uerid>,isGuest:true/false,isSet:true/false}. Calls next()
   5. Sets the response.header/response.cookie
6. API route receives the request.
   1. uses the request.user object to execute its logic
7. Client receives the response. Reads the response.header/response.cookie and sets the user in local storage/cookie

### Existing

1. User clicks "My Forms"
2. Client reads local storage/cookie for guest id
3. Calls server API:/ api/forms. Passes the guest id (cookie/header)
4. Request reaches server.
5. Auth middleware receives the request (Auth middleware is configured before all API routes)
   1. Reads the userid (request.cookie or request.headers)
   2. Sets request.user = {user:<uerid>,isGuest:true/false,isSet:true/false}. Calls next()
   3. Sets the response.header/response.cookie
6. API route receives the request.
   1. uses the request.user object to execute its logic
7. Client receives the response. Reads the response.header/response.cookie and sets the user in local storage/cookie

## Token-based Authentication

In token-based authentication, users register accounts and need to “Login” to the application.

1. Client (user) enters username/password.
2. Server validates username/password. If not valid, returns 401.
3. If valid, server creates an “Auth Token” and sends to client
4. Client stores the “Auth Token” and logs in the user.
5. Client sends “Auth Token” in each request to server (while the user is logged in)

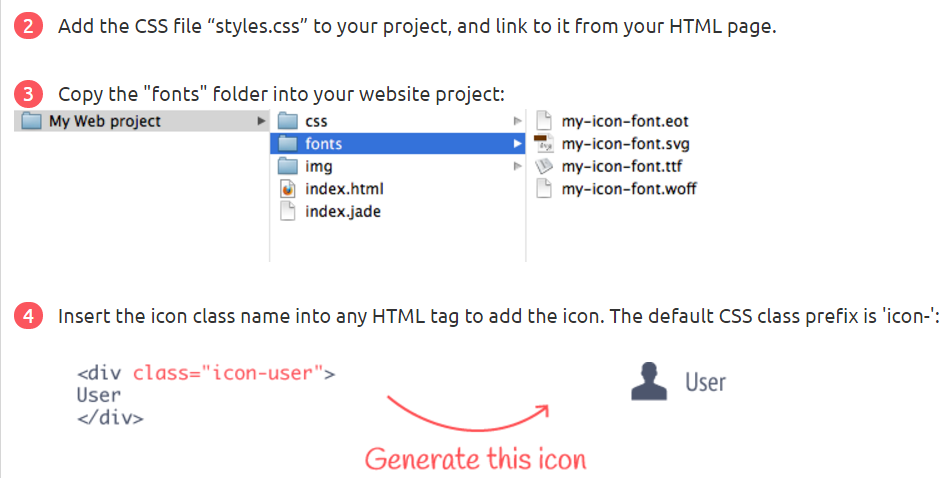
# Angular Mysteries

1. Sencha package equivalent in Angular. How to create a reusable library in Angular?
2. Understand and POC for Angular Universal
3. Writing code comments
4. AOT Compilation
   1. When doing AOT compilation using ng build command, faced build issues. The errors were about using private,protected members from the component template. Refer to the following post for details: <https://github.com/angular/angular/issues/11978>

# Custom Icon Font

One can create custom icon fonts using fantastic. Below are the steps:

1. Create an account on Fontastic.me.
2. Select the icon fonts from the list
3. Download. Following files are provided as part of the download:
   1. Css
   2. Reference HTML
   3. Font files (ttf,woff, etc)
4. Steps to integrate into the app



1. In Form Builder, following steps :
   1. In \_icon.scss , copy the content of css
   2. Copy fonts folder in a new font folder under styles

# Deployment

This section describes the deployment architecture, steps and technologies used.

## Building FormDesigner

### Auto Build

Run “Build Latest.bat”.

### Manual Build

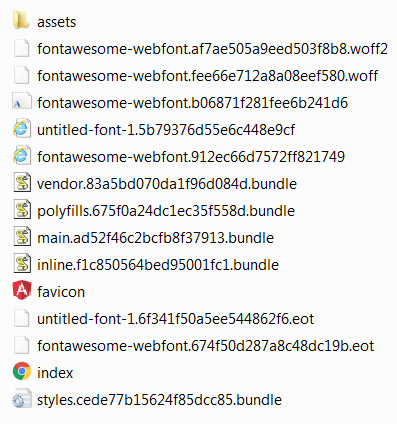
The first step in deployment is to do a production build of Form Designer Angular app. Angular CLI is used for doing a production build. Here are the steps:

1. CD into FormDesigner directory
2. Type : npm run build

NOTE: In package.json, “build” task is defined as:

"build":"ng build --aot true --output-path ../../../build/FormDesigner --target –prod --env=prod"

1. On successful build, the app artifacts will be copied to the output folder as shown below:



## Serving FormDesigner from Node

Node-based App Server needs to serve the FormDesigner Angular app. Here are the steps:

1. Create a folder called public -> FormDesigner under project root.
2. Copy the contents of FormDesigner to the folder.
3. Setup the static asset route using \_app.use(express.static('public/formDesigner'));
4. User can now access the app : http(s)://<server>:<port>/index.html

## Deploying MongoDB using MongoDB Atlas

The application uses MongoDB as a persistent storage solution. Instead of managing the MongoDB deployment on my own, we selected MongoDB Atlas as the preferred solution. MongoDB Atlas is a Database-as-a-Service product from MongoDB which manages the infrastructure on our behalf. We are currently using the Free Tier which deploys using AWS (use-east-1 region). Once the cluster is deployed, we need to configure the connection string in Node (in config.js)

## Deploying Node App Server

There are various ways of deploying Node-based web tier. For eg: using any of the cloud providers such as Azure, Heroku, AWS, etc. Since our MongoDB deployment is provisioned on AWS, we have selected AWS as the preferred cloud provider.

AWS provides different choice of services to meet our requirements. For eg: we can go PaaS way using AWS Elastic Beanstalk, or IaaS way using AWS EC2 and managing the instance ourselves.

### Using AWS Elastic Beanstalk

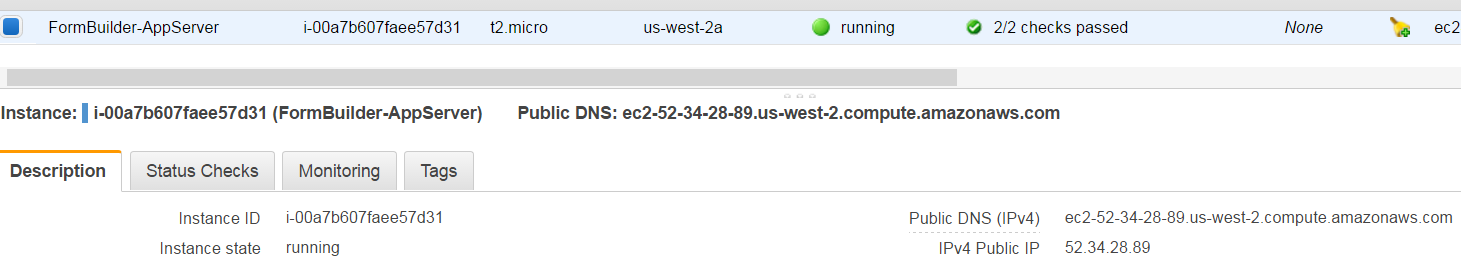
With Elastic Beanstalk, you can quickly deploy and manage applications in the AWS Cloud without worrying about the infrastructure that runs those applications. Elastic Beanstalk supports applications developed in Java, PHP, .NET, Node.js, Python, and Ruby, as well as different container types for each language.

Our Node app heavily uses async/await feature which was introduced in Node v7.6.0. Our current Node version is v7.9.0. AWS Elastic Beanstalk support only upto Node v6.1.0. Therefore, this option is ruled out in our case. We will keep a watch on the updates and will eventually move to deploying the app on Beanstalk.

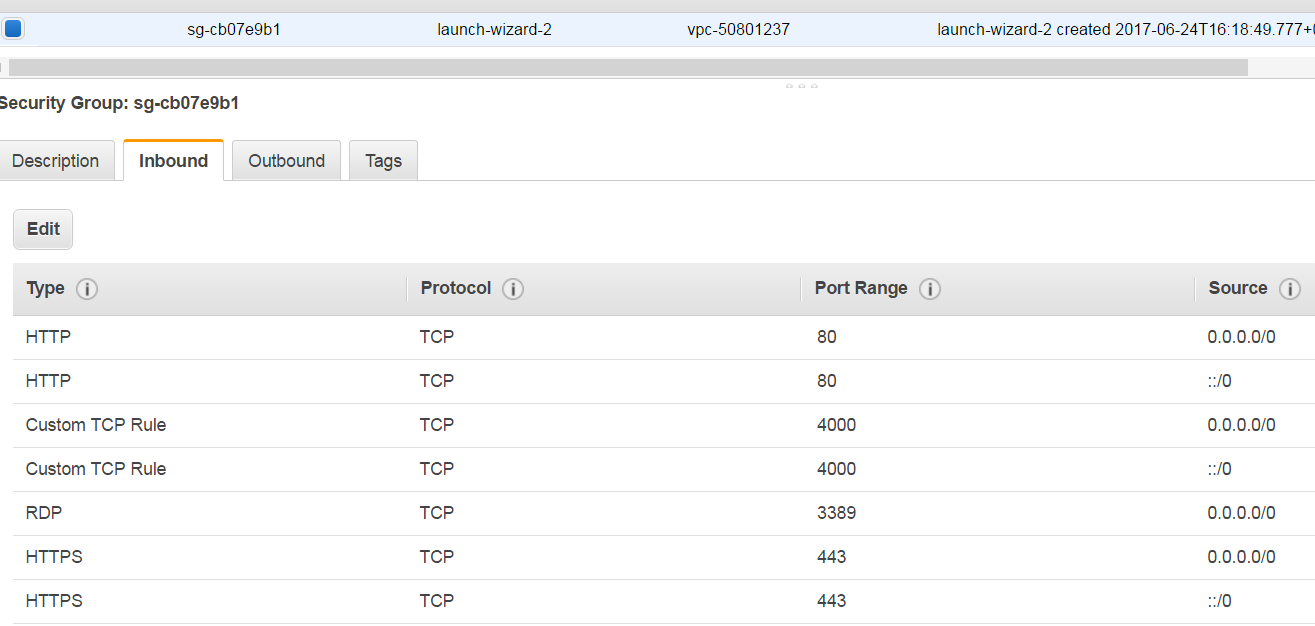
### Using AWS EC2

With Beanstalk out of the option, the Node app is now installed and deployed using AWS EC2 instance.

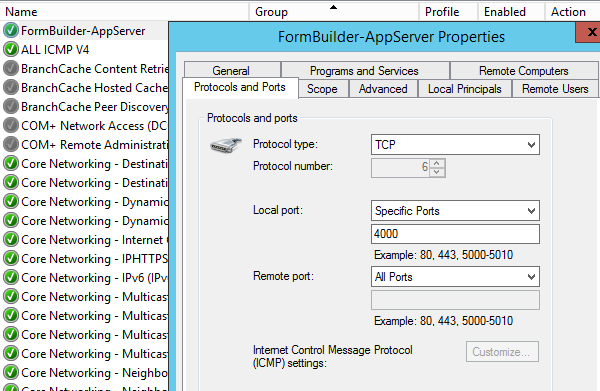
1. Launch a new EC2 instance (Windows server 2012 R2 Base)



1. Ensure Security groups allow Inbound and Outbound Traffic



1. RDS into the EC2 instance
2. Install Node v7.9.0
3. Zip the FormBuilder.Server code and copy to the EC2 instance
4. Open command prompt and type:
   1. Npm install
   2. Npm start
5. This should start the Node server running at port 4000/80.
6. Create a firewall rule to allow inbound traffic at port 4000/80.



## Using Cloud Foundry

Cloud Foundry is an PaaS provider.

### Getting Started

#### Installing CF CLI:

* https://docs.cloudfoundry.org/cf-cli/install-go-cli.html#windows

#### Creating account on run.pivotal.io

* Created account: email : furqan\_shaikh\_1999@hotmail.com, password: normal password with special characters

Created an org : FurqanOrg1

### Changes made to deploy to Cloud Foundry

#### Using Environment Variable

PCF has the concept of Services. MongoDb is one of the services available on the marketplace. When running the app as a cloud-native application, one of the design factors is to store config in the environment. Therefore, in FormBuilder server, we are reading the Mongo URI using the environment variable as shown in the code below:

function getMongoUrl(){

// If deployed to PCF, use the environment variable

var vcap = process.env.VCAP\_SERVICES

if(vcap){

let vcap\_services = JSON.parse(vcap)

return vcap\_services.mlab[0].credentials.uri;

}else{

return config.mongodb.url;

}

}

<https://docs.run.pivotal.io/buildpacks/node/node-service-bindings.html>

NOTE: For details on how to add service and bind to app, refer to Cloud Foundry documentation

#### Deploying Changes

1. Create a space in Org (Created Development space under FurqanOrg1 org)
2. Create manifest.yml and place under src folder

applications:

- name: FormBuilder-Dev

memory: 64MB

instances: 1

buildpack: https://github.com/cloudfoundry/nodejs-buildpack

command: npm start

1. Provide specific Node and NPM versions in package.json

"engines":{

"node":"8.5.0", -> This can vary based on current supported version

"npm":"4.2.0"

}

1. Change the serverUrl in environment.prod.ts to point to Cloud Foundry URL
2. Run “Build Latest.bat”
3. Open Command prompt and CD to source directory for Node server
4. Type cf push formbuilder-dev.
5. Voila!!! (<https://formbuilder-dev.cfapps.io>)

## Notes

1. Issues with AOT compilation
   1. If the template is referring to private/protected members of the Component class, AOT build will fail. Refer to for details : <https://github.com/angular/angular/issues/11978>
2. Issues with MongoDb Atlas
   1. This is a general issue. How do we set the connection string for different environments in Nodejs app. Currently we set the connection string in config.js. Before deploying to production, the value is changed manually.
   2. The password is stored in clear text. How do we encrypt ?
   3. If the password contains “@” , the connection fails. This is because the connection string uses “@” as a delimiter.
3. How to configure environment settings in Angular
   1. Server URL – Using environment
   2. Base href
4. Handling CORS issue
5. Provide the build pack path in manifest.yml to indicate to Cloud Foundry that this is a Node app.
6. Provide Node and NPM versions in package.json. If we don’t provide it, Cloud Foundry will pick some Node version. If you must target a specific version, it is recommended to provide version in package.json
7. Looks like the Node cluster doesn’t work on Cloud Foundry. So changed the start script to:

node infrastructure/server.js instead of node infrastructure/server.js

# Performance Improvements

## Form Builder Server

Server is built using Node v7.9.0 and Express web framework. Server provides:

1. Set of REST-based APIs
2. Authentication
3. Serving Form Designer Client application as static public assets

### Initial Perf Test

As part of initial perf test, /myforms API was tested.

**Tool:**

Apache Bench – A CLI tool for load testing

ab -r -n 100000 -c 1000 <http://localhost:4000/api/forms/0>

-n = Total number of requests

-c = concurrent requests

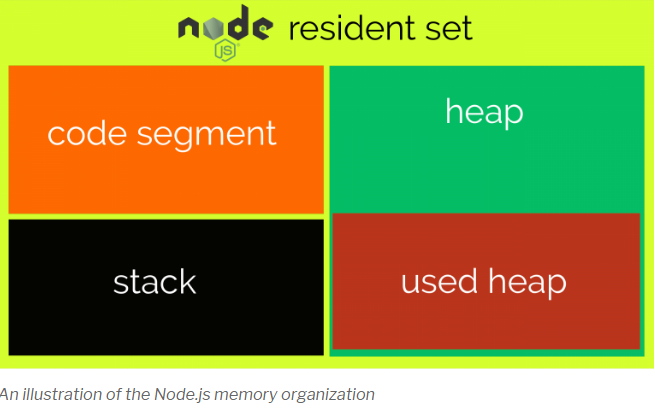
#### Output Results:

|  |  |  |
| --- | --- | --- |
|  | Db Connect per request | Improved Code (Db Connect on app start) |
| Failed Requests | 0 | 0 |
| Requests/Sec | 189 | 779 |
| Time/Request | 527 ms | 128 ms |
| Transfer Rate | 111.64 KB/sec received | 483 KB/sec received |
| Memory (Heap Used) | 460 MB | 29.18 MB |

#### As can be seen from the above results, the first test produced very poor results. The reasonis that the implementation was creating the MongoDb Client connection on every API request.

#### The fix was to make the MongoDb Client connection at the app start and cache it for later purposes.

### Nodejs Memory Model



<https://www.valentinog.com/blog/memory-usage-node-js/>

### Angular Performance

1. Always do production build and aot when shipping angular app

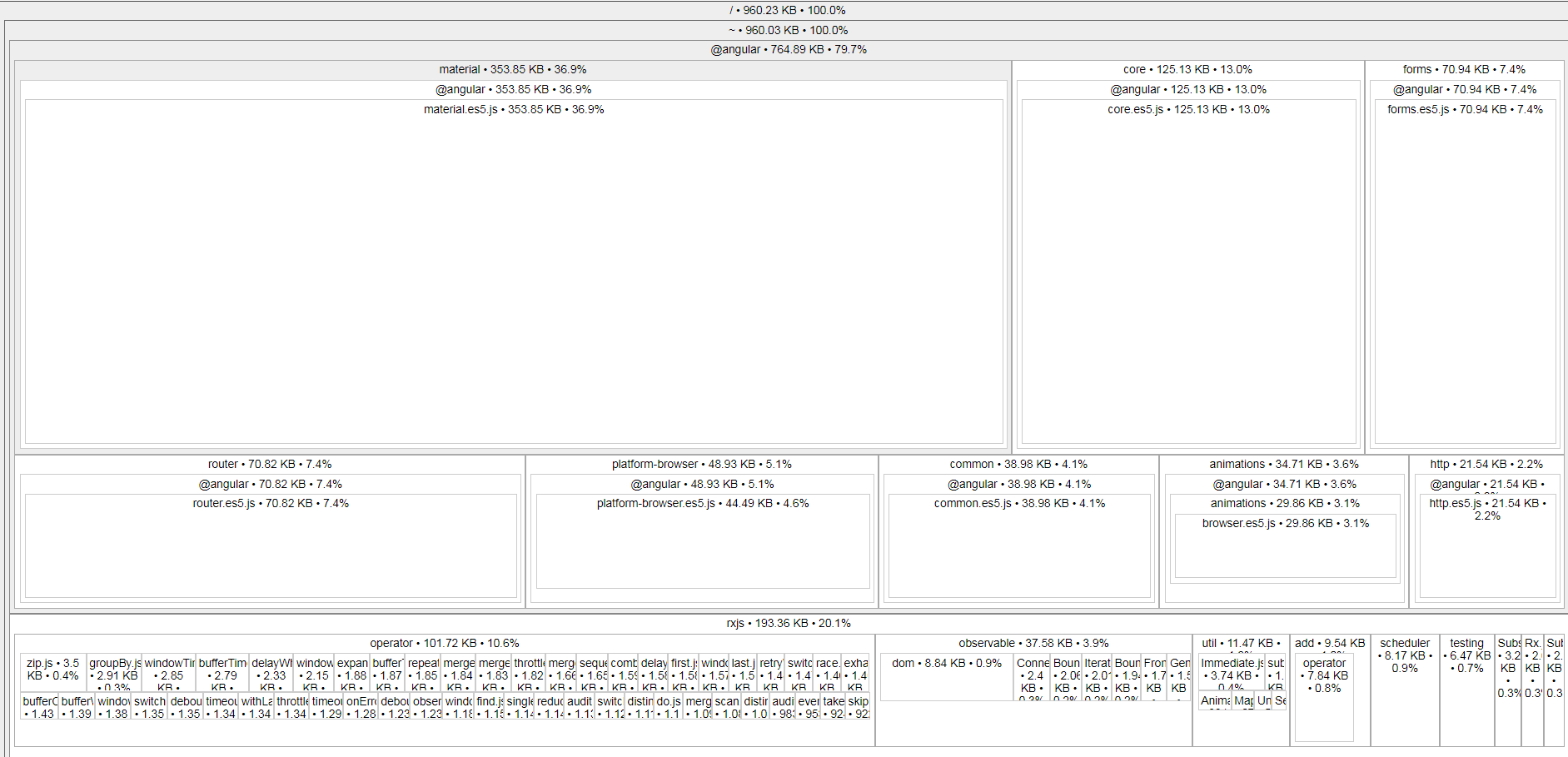
ng build --aot true --target –prod

AOT => Ahead-of-time-compilation

1. Reduce the application size by examining the build output. This can be done by using tools like Source Map Explorer. SME will provide great stats about how each module is contributing to the overall application size. One can examine the output and figure out unused code which can be removed to reduce the app size.

STEPS:

1. Npm install –g source-map-explorer
2. ng build --target -prod -sm
3. CD to output directory
4. Type : source-map-explorer vendor\*
5. It will open browser:



### Takeaway:

* Understanding of Nodejs memory model
* How to monitor memory usage using process.memoryUsage
* Using Apache Bench (ab) tool for load testing
* Efficient use of MongoDb Client in a Nodejs app

## Form Designer Client

## General

# Workflow – User

Store User Submissions

Published Forms



# Workflow - Design

Save

Create Form

Publish

Form Builder

Preview

View Submissions

Edit Form

List Forms

My Forms

Landing Page