



Ionic 2 Workbench documentation

Thanks for buying the Ionic 2 Workbench for Windows! Building apps with standard web technologies just became easier!

We know for sure it will be a big help while developing Ionic 2 applications. The workbench will do a lot of plumbing for you and will help you in finding & making the correct settings.

Contents

Installation prerequisites	3
Installing the Ionic 2 Workbench	4
Starting the workbench	5
Create an Ionic 2 project.....	6
Opening an Ionic 2 project.....	7
Using Ionic 2 controls.....	9
Example: drop the "Action sheet" control on a Typescript file	10
Using the property editor	12
Adding a new page or provider.....	13
Entity editor	14
Background	14
Create an entity	15
Form builder.....	17
Form Builder - General tab	19
Form Builder - Elements tab	20
Form Builder – Html preview tab.....	21

Installation prerequisites

First of all, make sure that you have the latest versions of:

- Node.js
Download from: <https://nodejs.org/>
- Ionic and Cordova
Open a command prompt and execute:
`> npm install -g ionic cordova`

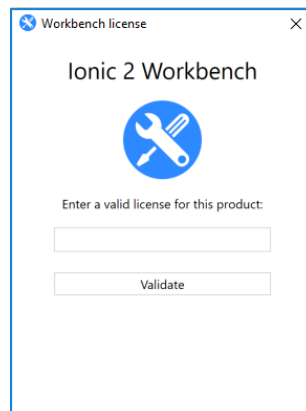
You can also check the "Installing Ionic" topic on the official website for further help:

<https://ionicframework.com/docs/v2/intro/installation/>

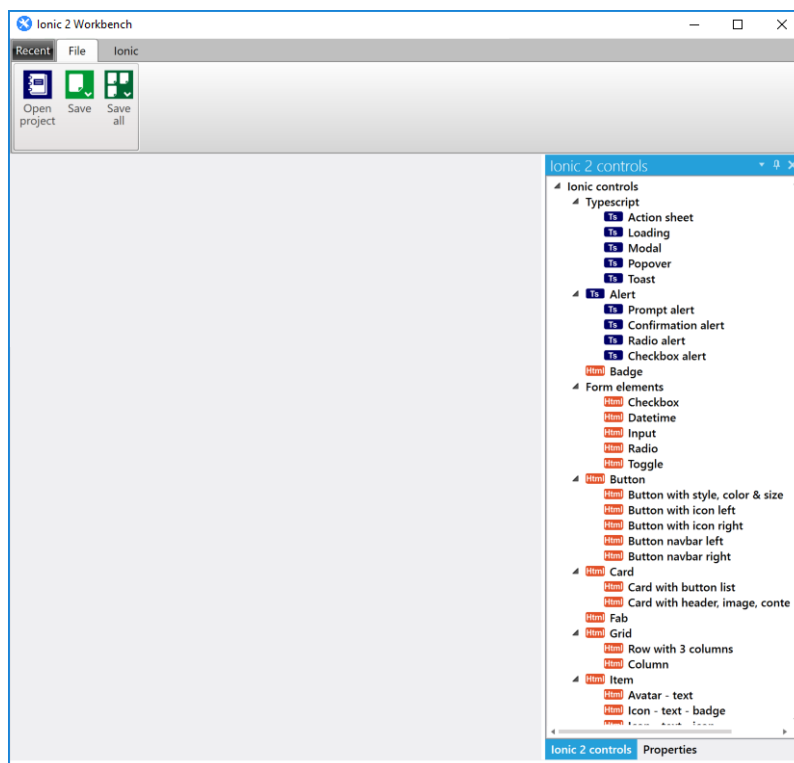
Installing the Ionic 2 Workbench

After purchasing the Workbench from the Da Vinci IT website (<http://dvit.eu>) you will receive a mail that contains an installation link and a license key. The installation will start after you click the link.

When installed you will see the license popup where you should input the license received after purchasing:



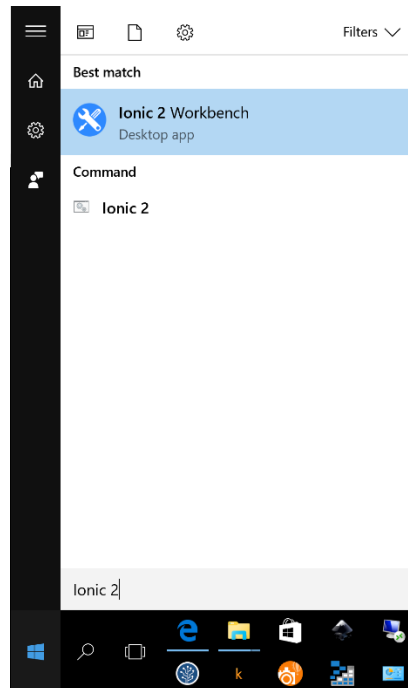
The workbench will start after the license is successfully validated:



From here you can open an existing Ionic 2 project or you can create a new project (see next topic).

Starting the workbench

Once installed you can start the workbench by pressing the Windows key and typing "Ionic 2".



For quicker access you can pin the Workbench to your start menu or taskbar.

Create an Ionic 2 project

In order to use the Workbench, you will need to have an existing ionic project.

If you don't have a project yet, then you can create a project using the command line (replace 'myApp' with a more appropriate name for your project):

```
> ionic start myApp --v2
```

```
C:\Code\sandbox\ionic2>ionic start myApp --v2
Creating an Ionic 2.x app in C:\Code\sandbox\ionic2\myApp based on the tabs template.

Downloading: https://github.com/driftyco/ionic2-app-base/archive/master.zip
Downloading: https://github.com/driftyco/ionic2-starter-tabs/archive/master.zip
Installing npm packages (may take a minute or two)...
|
🎉 🎉 🎉 Your Ionic app is ready to go! 🎉 🎉 🎉

Some helpful tips:

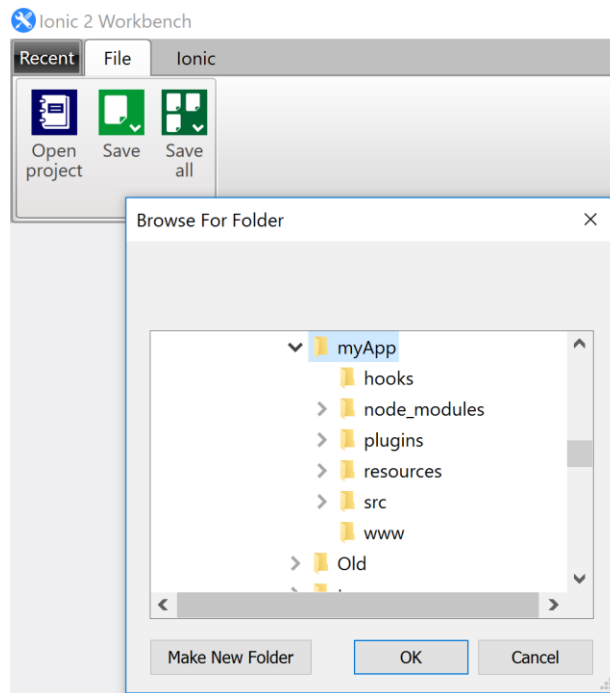
Run your app in the browser (great for initial development):
  ionic serve

Run on a device or simulator:
  ionic run ios[android,browser]

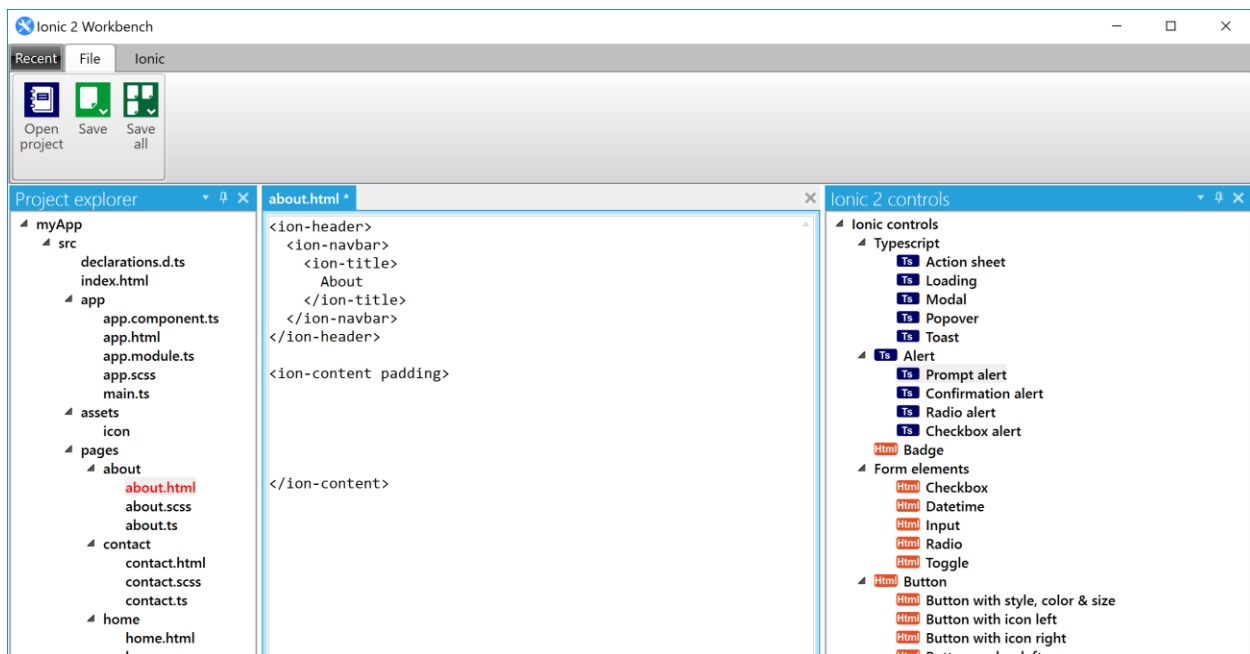
Share your app with testers, and test on device easily with the Ionic View companion app:
  http://view.ionic.io
```

Opening an Ionic 2 project

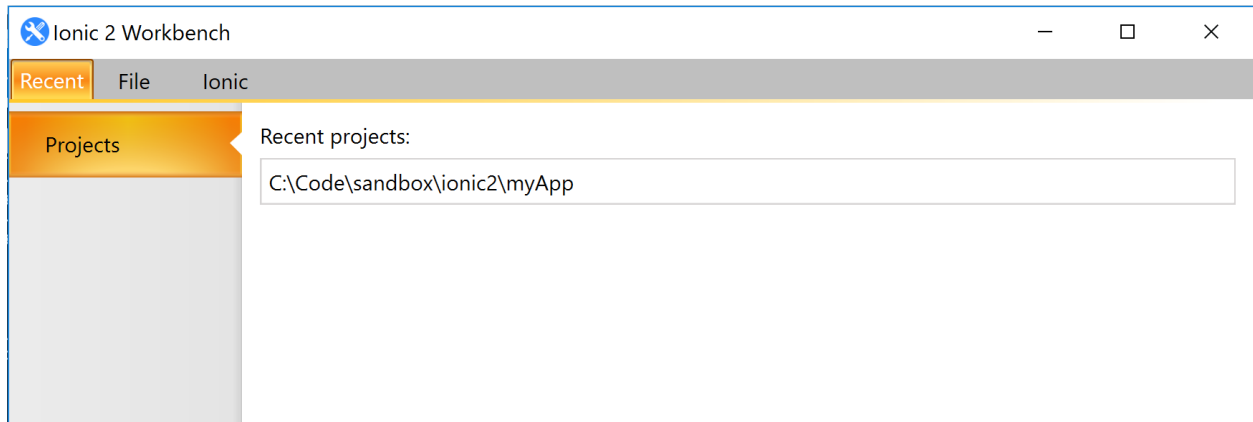
Press the "Open project" button and select the top level folder where your project is located. In our example we select the folder "C:\Code\sandbox\ionic2\myApp":



The workbench will open with your project structure on the left side:



The next time you open the workbench you can open previously opened projects more quickly by selecting it from the "Recent" tab:



Using Ionic 2 controls

The workbench will help you with writing all necessary code and wiring up things when integrating controls. With 1 drag & drop operation you can get a control to work instead of editing yourself on 3 or more places.

The workbench supports 2 types of controls:

Typescript controls:

These controls are prefixed with the typescript icon:



Be sure to drag & drop these controls on Typescript files.

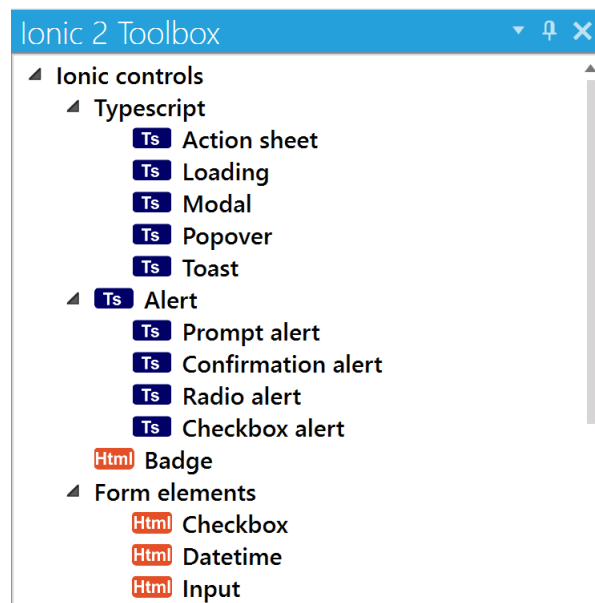
Html controls:

These controls are prefixed with the icon:

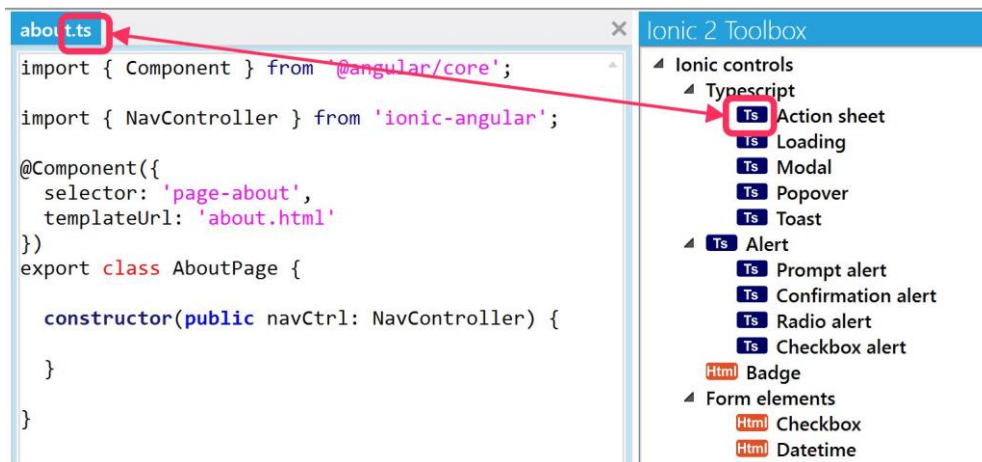


You should also drop these controls on html files.

Examples:



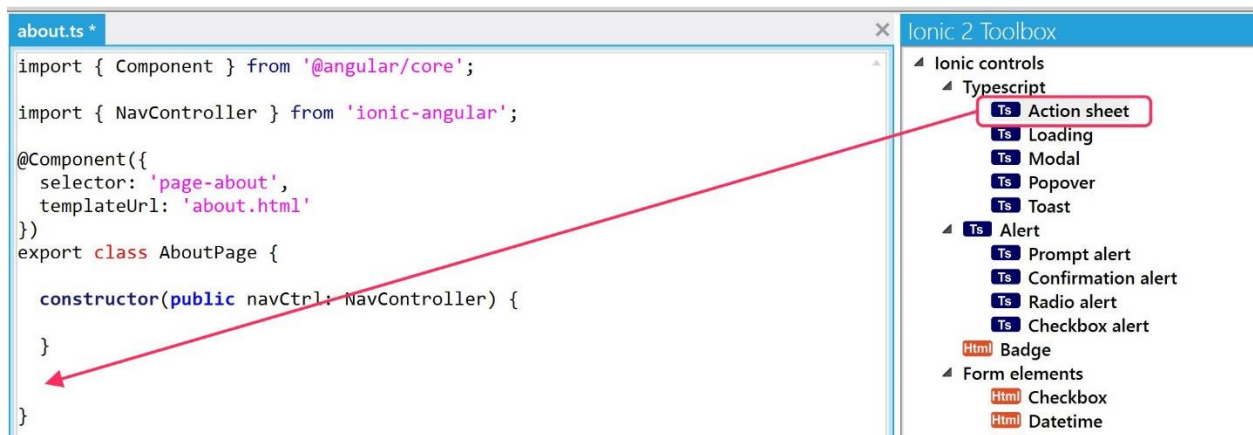
Be sure to drop on the correct file type. The icon of the control in the toolbox should match the file type:



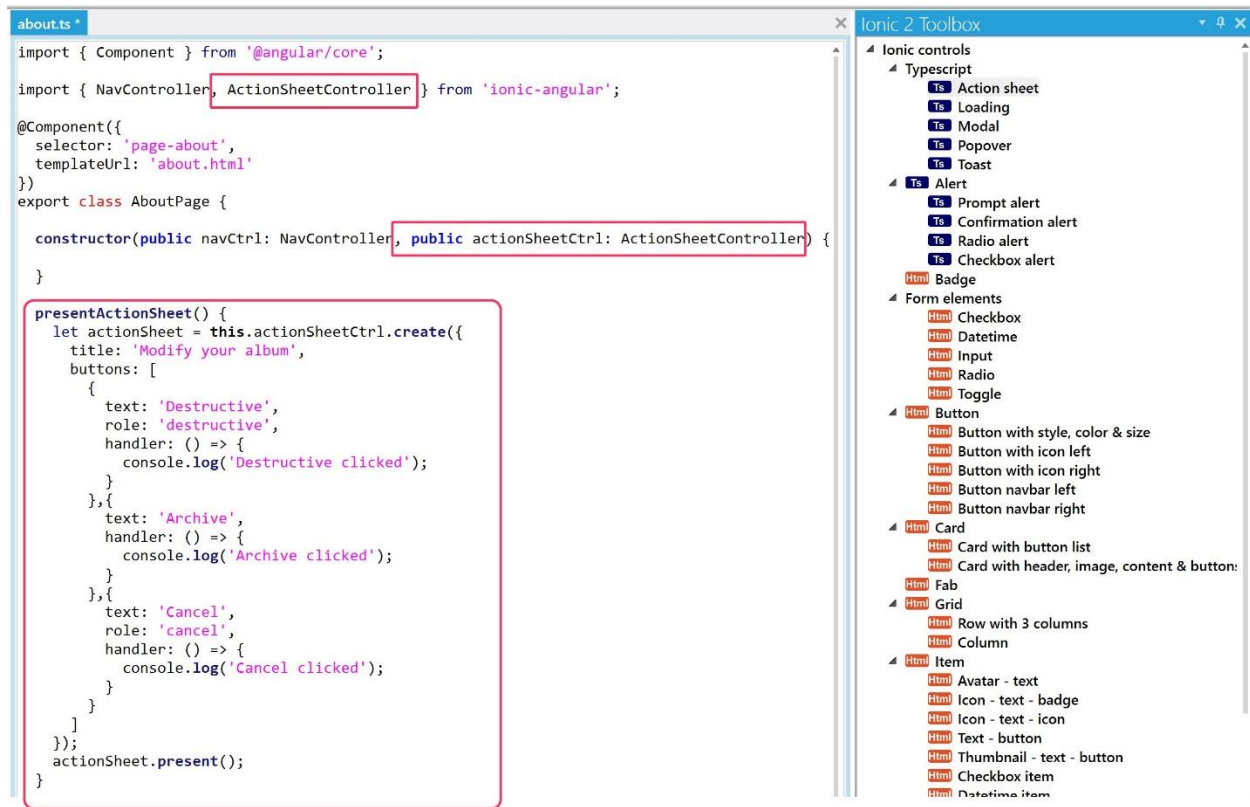
Example: drop the "Action sheet" control on a Typescript file

First, let's open any typescript file. Now drag & drop the "Action sheet" control into your code file.

Keep in mind that the control will be dropped on the exact location in your code file where you do the drop. Since most of the typescript controls will add a method (on top of adding the necessary imports and updating the constructor) it is best to drop them after the constructor where you would add new methods yourself:



This is the new situation after you dropped the control:



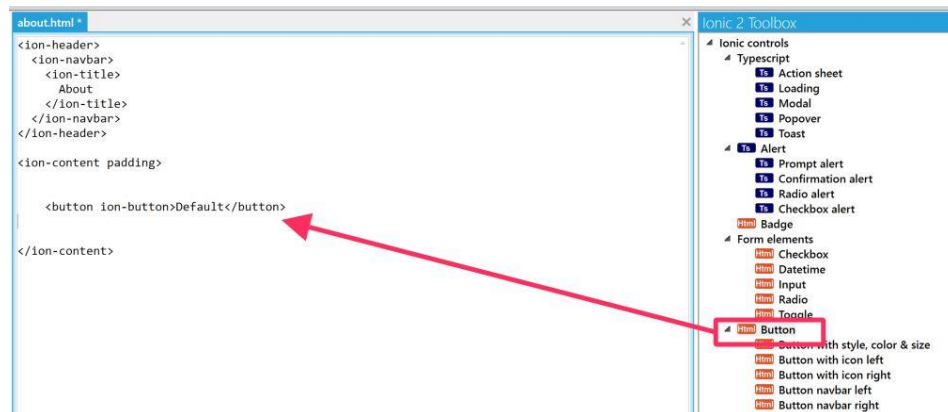
Dropping the control caused 3 changes to happen in your code file:

1. the `presentActionSheet()` method was added
2. the `ActionSheetController` import was added
3. the `'actionSheetCtrl'` property was injected via the constructor

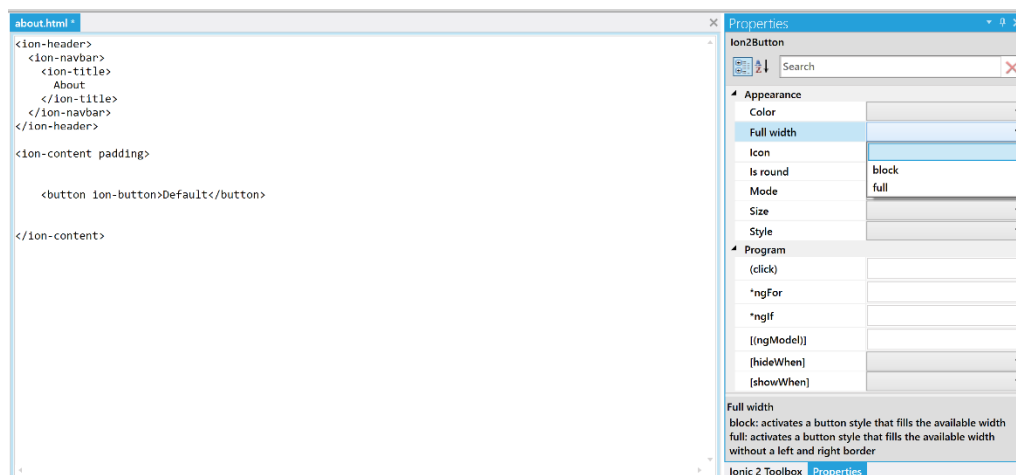
All the other controls will do similar actions in order to have them working.

Using the property editor

In order to demonstrate we will drop a basic button on a html file:



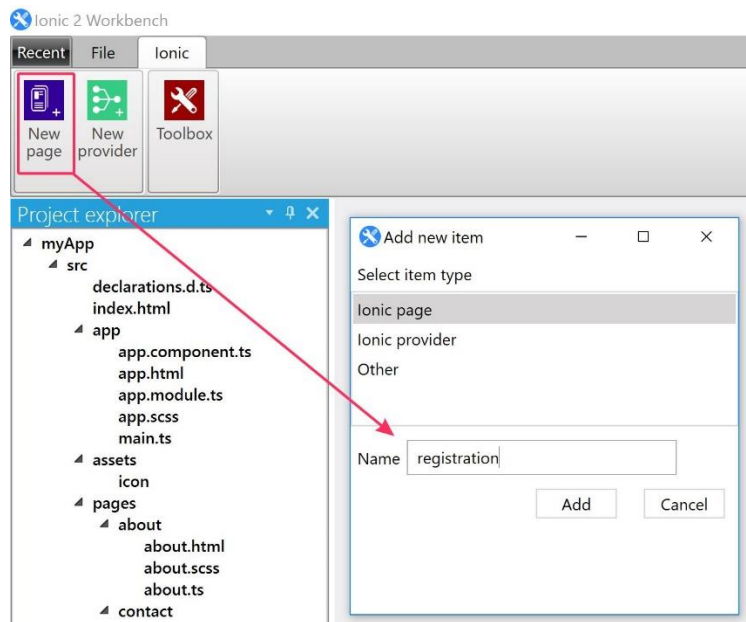
When you select the button html tag in the code editor and when you open the properties pane on the right side, you will see all possible settings for a button:



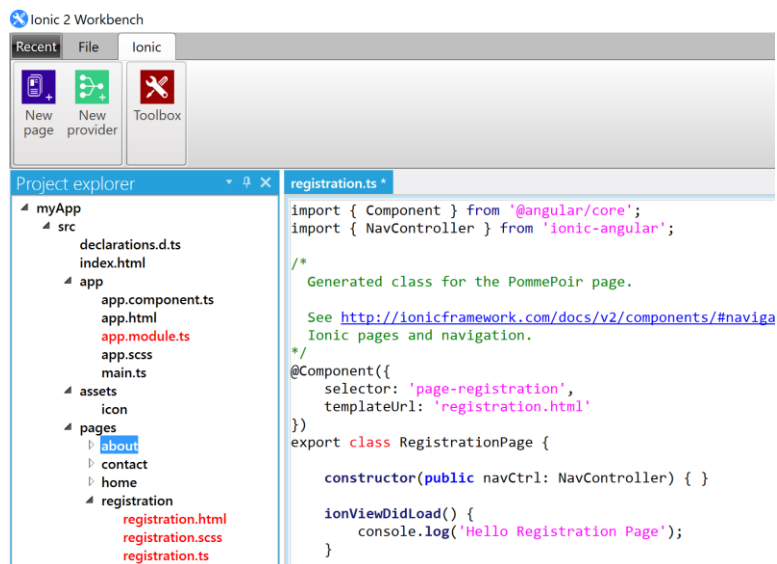
Now you can set any property and it will automatically update your code. Also notice the documentation of the property that is available on the bottom right side.

Adding a new page or provider

You add a new page simply by pressing the "New page" button in the Ionic menu. Next you type in the name and press add:



As a result, you will get:



On the left side you will see that all necessary pages were added (html, scss, ts) and that the "app.module.ts" file was updated.

Entity editor

Background

When you introduce a new object in your application you will most likely need to write code on several different locations throughout your code base.

It's not exceptional to find code on 6 or more different locations in your development projects referring to this object.

In enterprise applications you can find for instance specific code for this object in:

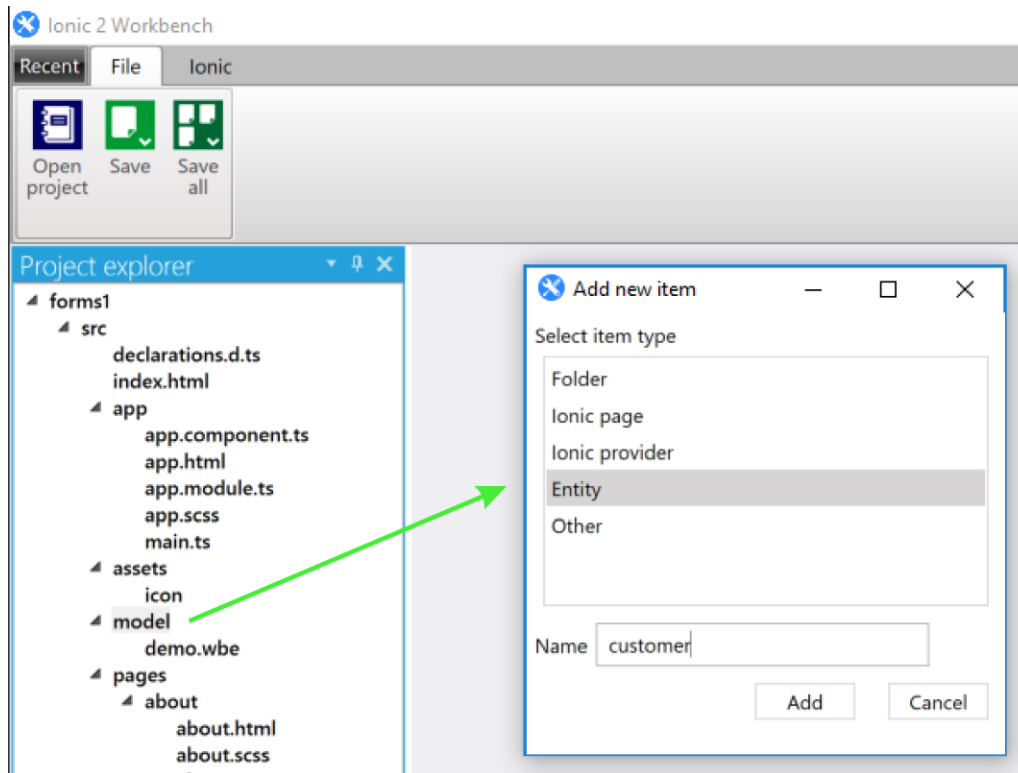
- the backend project:
 - class definition (in backend language such as .NET, Java, Perl, ...)
 - specific database code
 - web API
- the front end project:
 - class definition (Javascript, Typescript)
 - client library
 - forms and tables

This means that every new object requires a lot of work to setup. So in order to reduce the amount of work we first define our application objects independently of any programming language.

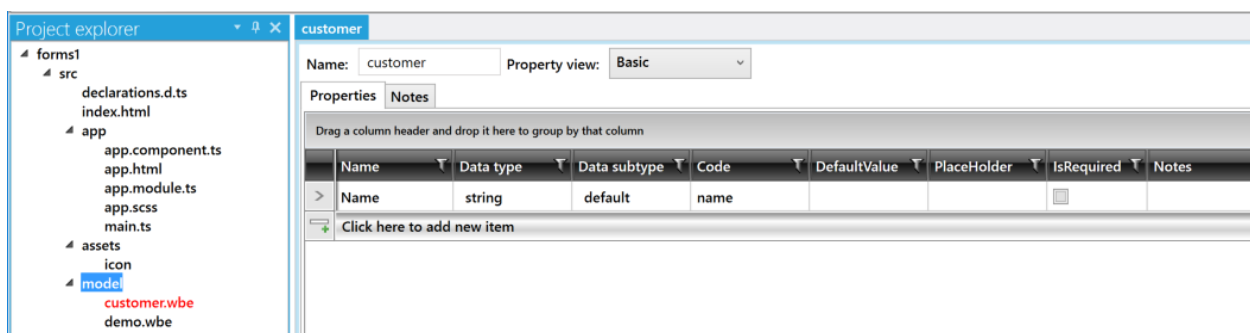
We use the Workbench entity editor for this purpose and we try to collect as much information as possible about this entity. This tool basically generates a file in JSON format containing all info about the object (file with extension .wbe) and this is the master definition for the object for the complete code base. All other code construct should be derived as much as possible (generated) from this definition.

Create an entity

Right-click on any folder inside your project and select “Add...”, then select the item type “Entity” and type the name of your entity (in this example ‘customer’):



The workbench will then create a new file ‘customer.wbe’ with already 1 property in it (the property ‘Name’ of type ‘string’):



Remark: this entity file has the extension ‘.wbe’ meaning **W**ork**B**ench **E**ntity. But it’s basically just a text file in json format that you can easily open and visualize with any text editor.

Now it's easy to create all the properties of your customer entity:

customer

Name: customer Property view: Basic

Properties Notes

Drag a column header and drop it here to group by that column

	Name	Data type	Data subtype	Code	DefaultValue	Placeholder	IsRequired	Notes
>	Name	string	default	name			<input type="checkbox"/>	
	Address	string	bigtext	address			<input type="checkbox"/>	
	Birthday	datetime	default	birthday			<input type="checkbox"/>	
	Female	boolean	default	isFemale			<input type="checkbox"/>	
	Nr of children	number	default	nrOfChildren			<input type="checkbox"/>	
	Password	string	password	password			<input type="checkbox"/>	

Click here to add new item

For each property you can configure the:

- Data type: string, number, boolean, datetime
- Data subtype: based on the type selection you will have more specific options:
 - Data type = string
 - default: use this for basic strings
 - bigtext: multiline text
 - email
 - number
 - password
 - tel
 - ...
 - Data type = number
 - Integer:
 - Money: to store monetary values
 - ...
- Code: this is the variable named to be used in the generated code for this property (the value is initially derived from the 'Name' you supplied, but you can change it).
- Default value: fill in when your property has a default value
- Placeholder: the place holder to be shown in the user interface for this property
- Is required: if the property is a required fill-in. Forms that will be generated will require that this field will be filled in before submitting for example.
- Notes: here you can enter some documentation for this property

And finally there is also a 'Notes' tab where you can leave some documentation about the entity. Also note that you can fill in more properties specific for strings, numbers, etc if you change the property view next to the entity name (settings such as minLength, maxLength, min, max, step, ...).

Form builder

Almost any app will have 1 or more forms requesting the user for information. With the form builder you can create custom forms in no time.

Form Builder

General Elements Html preview

Form Builder

Customer details

Name Frank Smith

Address Highway, 6 New York

Birthday Jan 1, 1978

☐ Female

Nr of children 2

Password ••••••••

Insert code

Ion2Form

Search

Appearance

Container type no container

Label format floating

Title customer

Misc

novalidate ☒

Programming

(ngSubmit) onSubmit()

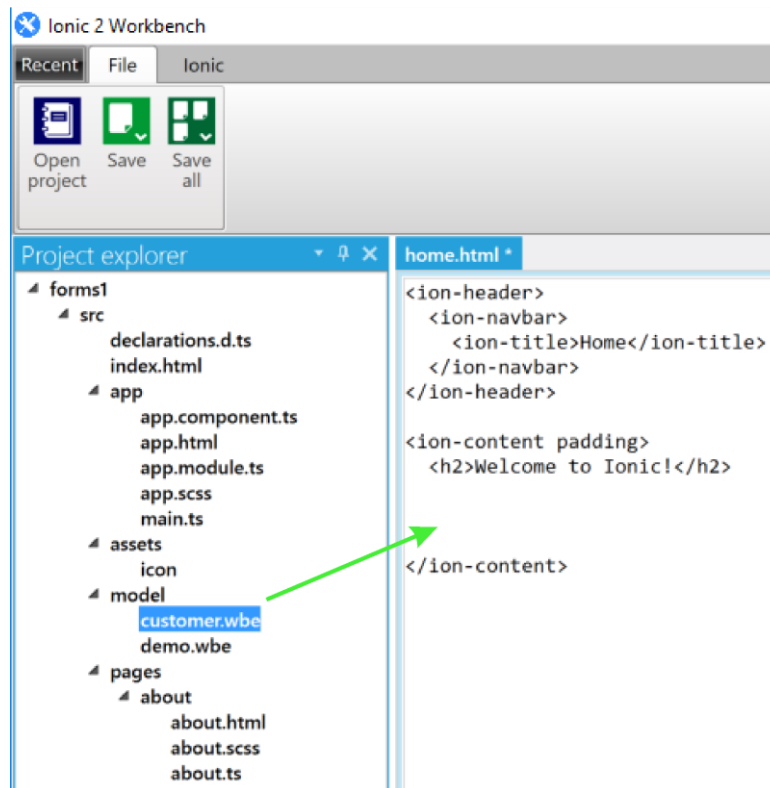
[(ngModel)] prefix

ngForm name f

In order to create a form, you will first have to create an entity (previous topic) that defines the elements of the form. Fill in for each property of your entity at least the name and the type.

Remark: The form that is visualized in the Form Builder window above was created automatically from the 'customer' entity.

Then you have to drag & drop this entity to the correct location in any html file of your project:



The form builder window will automatically appear. You have 3 tabs in the form builder.

Form Builder - General tab

The screenshot displays the 'Form Builder' application window. The 'General' tab is active, showing a preview of a form titled 'Customer details'. The form contains several input fields: 'Name' (Frank Smith), 'Address' (Highway, 6 New York), 'Birthday' (Jan 1, 1978), 'Gender' (Female), 'Nr of children' (2), and 'Password'. A green 'Insert code' button is located at the bottom of the preview area. To the right, the 'Ion2Form' configuration panel is visible, allowing users to customize the form's appearance and behavior. The 'Appearance' section includes 'Container type' (card), 'Label format' (floating), and 'Title' (Customer details). The 'Misc' section has 'novalidate' checked. The 'Programming' section shows 'ngSubmit' as 'onSubmit()', '[(ngModel)] prefix' as an empty field, and 'ngForm name' as 'f'. A note at the bottom of the configuration panel explains that the 'novalidate' setting is advised to bypass default HTML5 validation in favor of Angular 2 validation.

Here you can configure some general settings about your form. Most of the settings will also give extra documentation underneath the property grid. (in the example above you see that the setting “novalidate” is selected and at the bottom you see some extra information about this setting).

Extra information:

Container type	If you want the form to be embedded in another visual object, then you can specify this here.
Label format	Specifies how the labels should be displayed. For Ionic: the options are fixed, floating, inline, inset, stacked More information can be found here: https://ionicframework.com/docs/v2/components/#inputs
Title	The title to be displayed above the form
novalidate	It is advised to set the 'novalidate' in order to bypass the default HTML5 validation because you are using the Angular 2 validation instead.
[(ngModel)] prefix	The form elements will bind by default directly to root properties in your component (in the Typescript file). For example: the Name input field will bind directly to the 'name' property in your Typescript file. If you want to bind to properties inside a sub-object, then you can specify this here. So if you want to bind to 'customer.name' then you set this prefix to: 'customer.'
ngForm name	The form tag can have an attribute like #f=""ngForm". In this case you specify 'f' here. As a result, you can use f as the variable representing the Angular form in your code (for instance for validation purposes).

After you reviewed to other tabs (see also below), you should press the “Insert code” button to insert the form on the location where you performed the drop operation.

Form Builder - Elements tab

The elements will be automatically generated from your entity definition. It should be good enough to start with, but you can change whatever you want:

Form Builder ✕

General Elements **Html preview**

Drag a column header and drop it here to group by that column

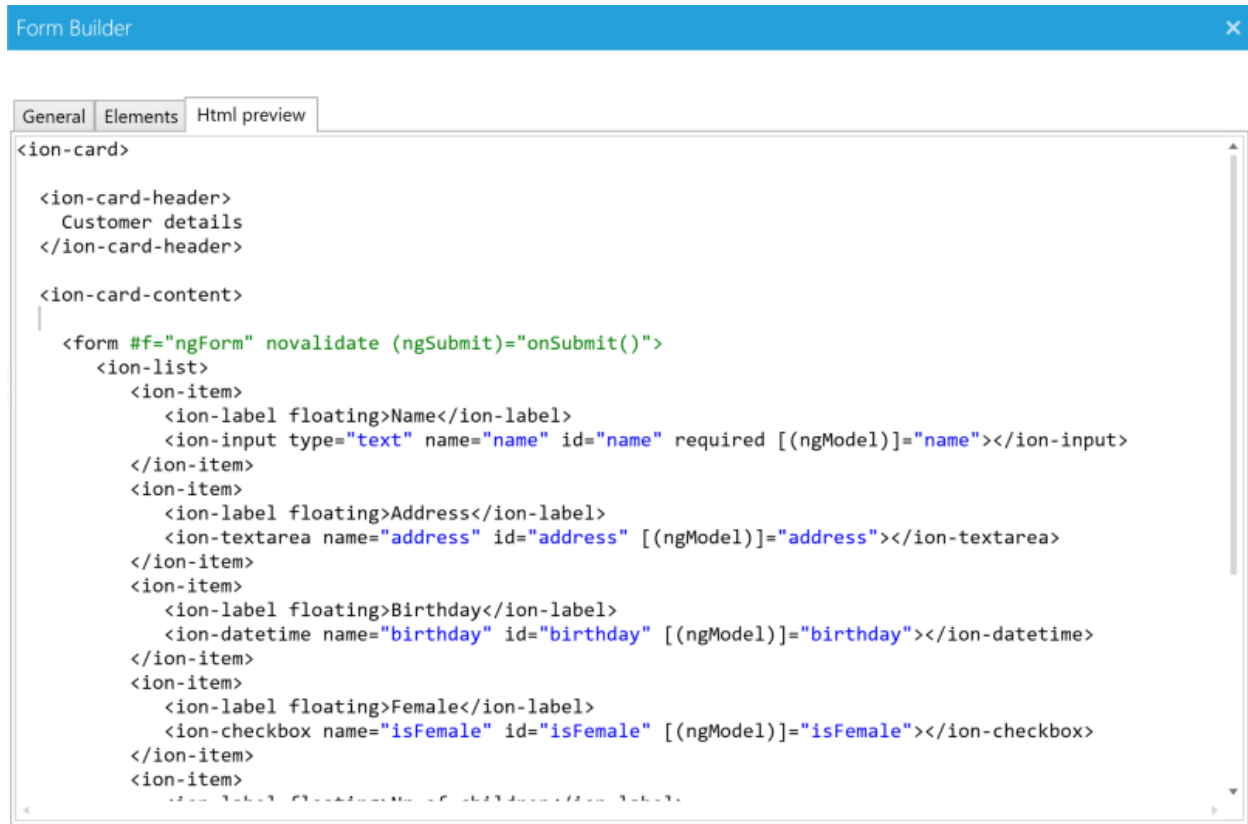
	Label	Name	Id	Control Type	NgModel	Placeholder	IsRequired	
>	Name	name	name	input-text	name		<input checked="" type="checkbox"/>	
	Address	address	address	textarea	address		<input type="checkbox"/>	
	Birthday	birthday	birthday	datetime	birthday		<input type="checkbox"/>	
	Female	isFemale	isFemale	checkbox	isFemale		<input type="checkbox"/>	
	Nr of children	nrOfChildren	nrOfChildren	input-number	nrOfChildren		<input type="checkbox"/>	
	Password	password	password	input-password	password		<input type="checkbox"/>	

Click here to add new item

Label	This is the label that will be shown to the end user.
Name	This will be used for the html 'name' attribute of the html control
Id	This will be used for the html 'id' attribute of the html control
Control type	<p>This can be changed via a drop down.</p> <p>For Ionic: input-text will end up in a html control <code><ion-input type="text"></code></p>
NgModel	This is the name of the property that this control will bind to in your TypeScript file. If you specified a prefix in the general tab ([ngModel] prefix), then the NgModel value that you specify here will be prefixed with that specific prefix.
Placeholder	The placeholder will be shown when the control is not filled in by the end user and will disappear when something is filled in.
IsRequired	Is required

Form Builder – Html preview tab

In the last tab you can preview the generated html:



```
<ion-card>
  <ion-card-header>
    Customer details
  </ion-card-header>
  <ion-card-content>
    <form #f="ngForm" novalidate (ngSubmit)="onSubmit()">
      <ion-list>
        <ion-item>
          <ion-label floating>Name</ion-label>
          <ion-input type="text" name="name" id="name" required [(ngModel)]="name"></ion-input>
        </ion-item>
        <ion-item>
          <ion-label floating>Address</ion-label>
          <ion-textarea name="address" id="address" [(ngModel)]="address"></ion-textarea>
        </ion-item>
        <ion-item>
          <ion-label floating>Birthday</ion-label>
          <ion-datetime name="birthday" id="birthday" [(ngModel)]="birthday"></ion-datetime>
        </ion-item>
        <ion-item>
          <ion-label floating>Female</ion-label>
          <ion-checkbox name="isFemale" id="isFemale" [(ngModel)]="isFemale"></ion-checkbox>
        </ion-item>
      </ion-list>
    </form>
  </ion-card-content>
</ion-card>
```