My Adventures Running Angular Ivy inside StackBlitz: Yes, It Is Possible!

Learn The Tricks I Used To Quickly Experiment with new Angular Rendering Engine

Uri Shaked Follow
Nov 27, 2018 · 7 min read

This post is the story of how I made Ivy, the new Angular renderer, run inside StackBlitz. Instead of just giving out the solution, I decided to share the process I went through, so you will also be able to learn how I approach these kind of challenges. But first, the most important question—why did I even bother?

A few months ago I ran Ivy and looked at the code it generates from your templates. Then, I found a way to write the generated code myself, "Do-It-Yourself" style, and even created a small StackBlitz demo that would run the manually crafted code and render the result to the screen.

It was all fun, but when Angular 7 was released, some of Ivy's internals were changed, and my hand-crafted code stopped working. Obviously, this was expected—Ivy is still under heavy development and I was using some internal, non-public APIs.

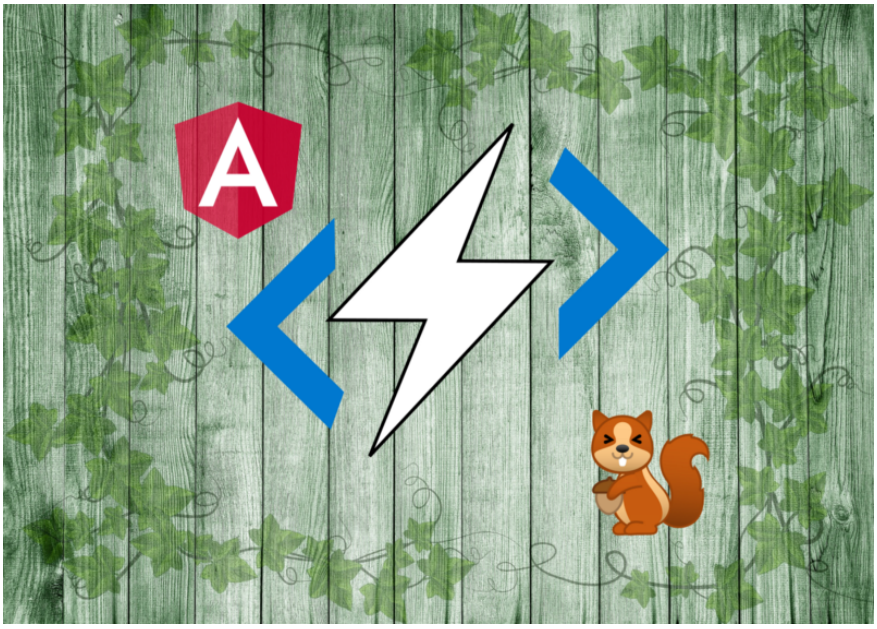
Netta Bondy and I are currently working on our "React Fiber vs. Angular Ivy" talk for FrontEnd Con in Warsaw (that's next week!), and we wanted a way to quickly prototype and test things with Ivy. StackBlitz is my favorite tool for these quick tests (also live-coding demos in my talks), and we used it extensively when trying to figure out various edge cases with React.

Naturally, we wanted to use StackBlitz with Ivy too, and without having to manually write the generated code for the template, like I did in the previous blog post—we wanted to have the Ivy compiler run inside the browser and do this for that. However, we quickly realized that:

StackBlitz Doesn't Support Ivy (Yet)

I totally understand why they don't support Ivy. As I mentioned above, the APIs are not final, and adding Ivy support would make them depend on internal APIs that are likely to break. I met Eric Simons a few weeks ago, and he told me about some of the amazing stuff they are currently building—so they are probably too busy to work on something that is bound to break.

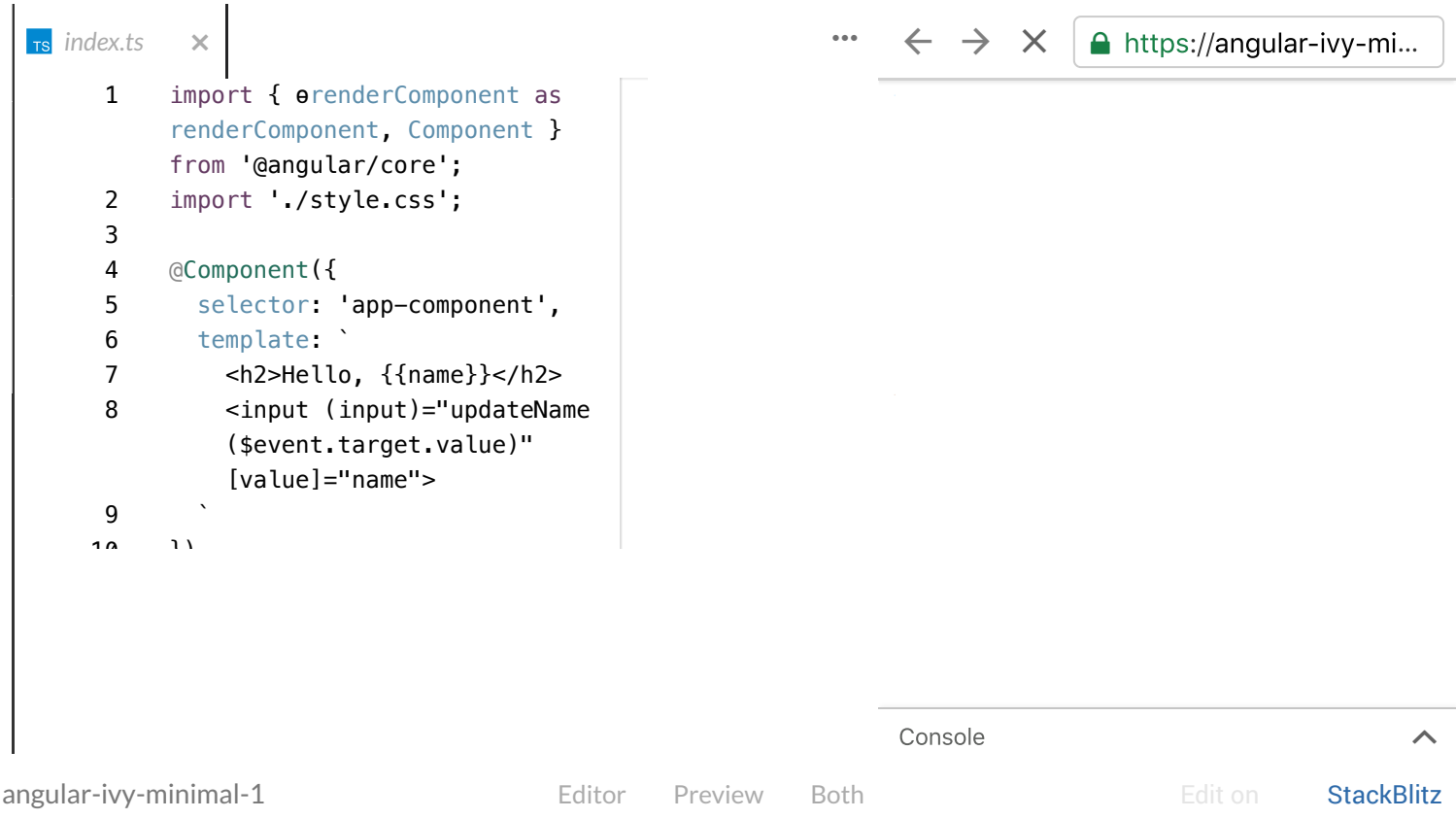
However, even if StackBlitz doesn't support Ivy, it doesn't mean it can't be done—and actually, it is quite easy. I'm going to walk you through the process, showing you what I tried and how I eventually got it to work. Let's start!



Ivy and StackBlitz—will they fit together?

Define a Component, Call renderComponent, hope for the best!

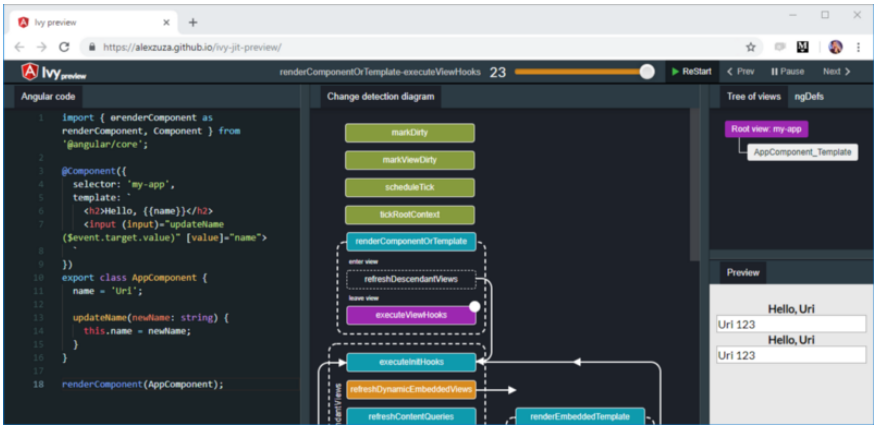
I started with a very naive solution: define a simple component using the `@Component()` decorator, and pass the class to `ɵrenderComponent`, an internal Angular method that renders an Angular component to the screen, hoping that it would work:



As you can see, this attempt failed miserably. The error message is basically Ivy telling us that it couldn't find the `ngComponentDef` property on our `AppComponent`. This property holds all the metadata for compiled Ivy components. This basically means that the Ivy compiler did not run inside the browser.

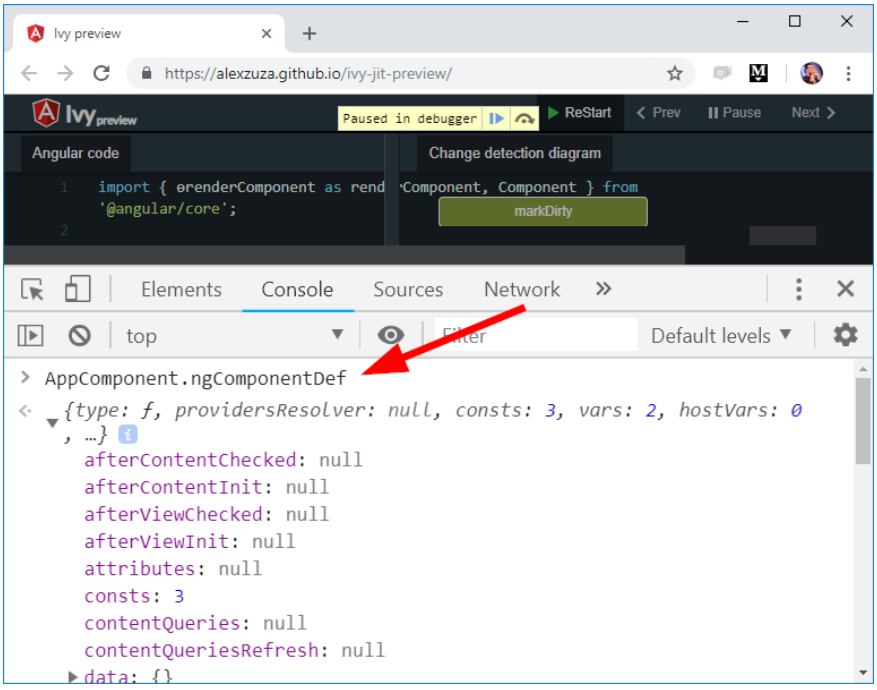
Ivy, Y U No Compile?

I scratched my head for a few minutes. I knew it was possible to get the compiler running in the browser, as I saw this awesome demo by Alexey Zuev, which managed to compile Angular templates using Ivy in the browser. I even pasted my own code there, hit the "Start" button—and it worked like a charm:



My code (with minor adaptations) worked perfectly inside this Ivy Preview playground

Obviously, I was missing something. I put a `debugger;` statement just before the call to `renderComponent`, opened Chrome Dev Tools and looked at the value of `AppComponent.ngComponentDef` in both apps (Alexey's app and my StackBlitz app). Interesting, in Alexey's environment, it actually had a value:



Whereas in my StackBlitz demo, it was simply `undefined`. There was something special about Alexey's environment that caused the `@Component` decorator to add this special property into the class. In other words, in Alexey's environment, `@Component` would run the Ivy compiler on the component, whereas in my environment it wouldn't.

Unfortunately, Alexey didn't open-source his playground, so I couldn't just look into the source and find the answer. So I went with

a different approach:

JavaScript Debugging Sorcery

I wanted to find what piece of code was setting the value of the `ngComponentDef` property in Alexey's environment. Since I assumed this piece of code would run inside the `@Component` decorator, I had to find a way to set a trap before this decorator code executed. Thus, I came up with the following game plan:

1. Create a new decorator, which I called `Trap`
2. Apply this decorator to the `AppComponent` class, so it runs just before `@Component` decorator (decorators are run in right-to-left order, so the decorator declared just before the `class` clause runs first)
3. Inside my decorator I used `object.defineProperty` to make the `ngComponentDef` property read only. This would ensure an error will be thrown whenever somebody tries to set this property—in this case, Angular's Ivy compiler. Then I will be able to look at the stack trace and see the code path the led to the invocation of the compiler

This is what the code looked like:

```
1  function Trap(clz) {
2    Object.defineProperty(clz, 'ngComponentDef', { writa
3    return clz;
4  }
5
6  @Component({
7    selector: 'my-app',
8    template: `...`
9  })
```

Yes, decorators are just regular JavaScript functions. I apply the decorator in line 10

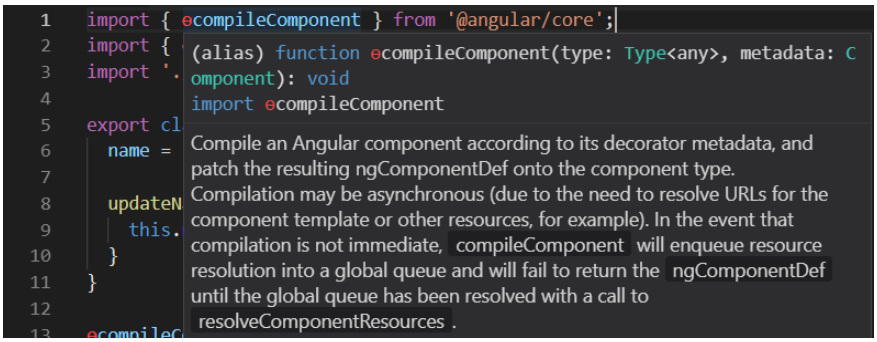
I hit the "Start" button again and got a nice error trace in the Preview pane:

```
TypeError: Cannot redefine property: ngComponentDef
  at Function.defineProperty (<anonymous>)
  at compileComponent (https://alexzuza.github.io/ivy-jit-preview/ngEntry.d83e06286282c874b82d.js:36862:12)
  at https://alexzuza.github.io/ivy-jit-preview/ngEntry.d83e06286282c874b82d.js:36786:65
  at TypeDecorator (https://alexzuza.github.io/ivy-jit-preview/ngEntry.d83e06286282c874b82d.js:10653:24)
  at https://alexzuza.github.io/ivy-jit-preview/main.d83e06286282c874b82d.js:16:7597
  at Object.<anonymous> (https://alexzuza.github.io/ivy-jit-preview/main.d83e06286282c874b82d.js:16:7662)
  at __decorate (ng:///ivy.js?joyzw8x6:8:92)
  at eval (ng:///ivy.js?joyzw8x6:25:20)
  at eval (ng:///ivy.js?joyzw8x6:33:2)
  at https://alexzuza.github.io/ivy-jit-preview/ngEntry.d83e06286282c874b82d.js:42028:61
```

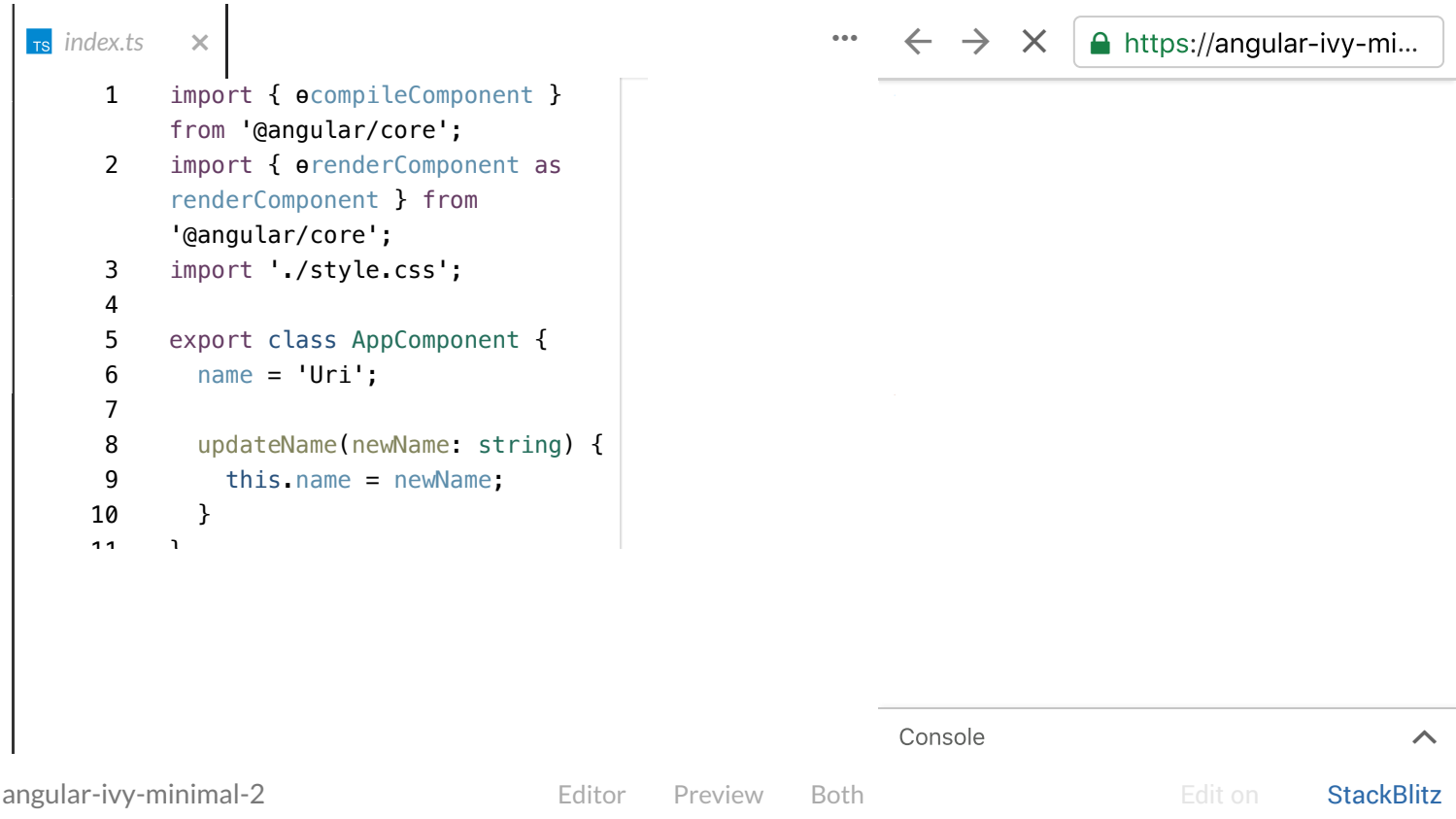
Voila! The method that sets the value for `ngComponentDef` is called `compileComponent`!

From one Error to another

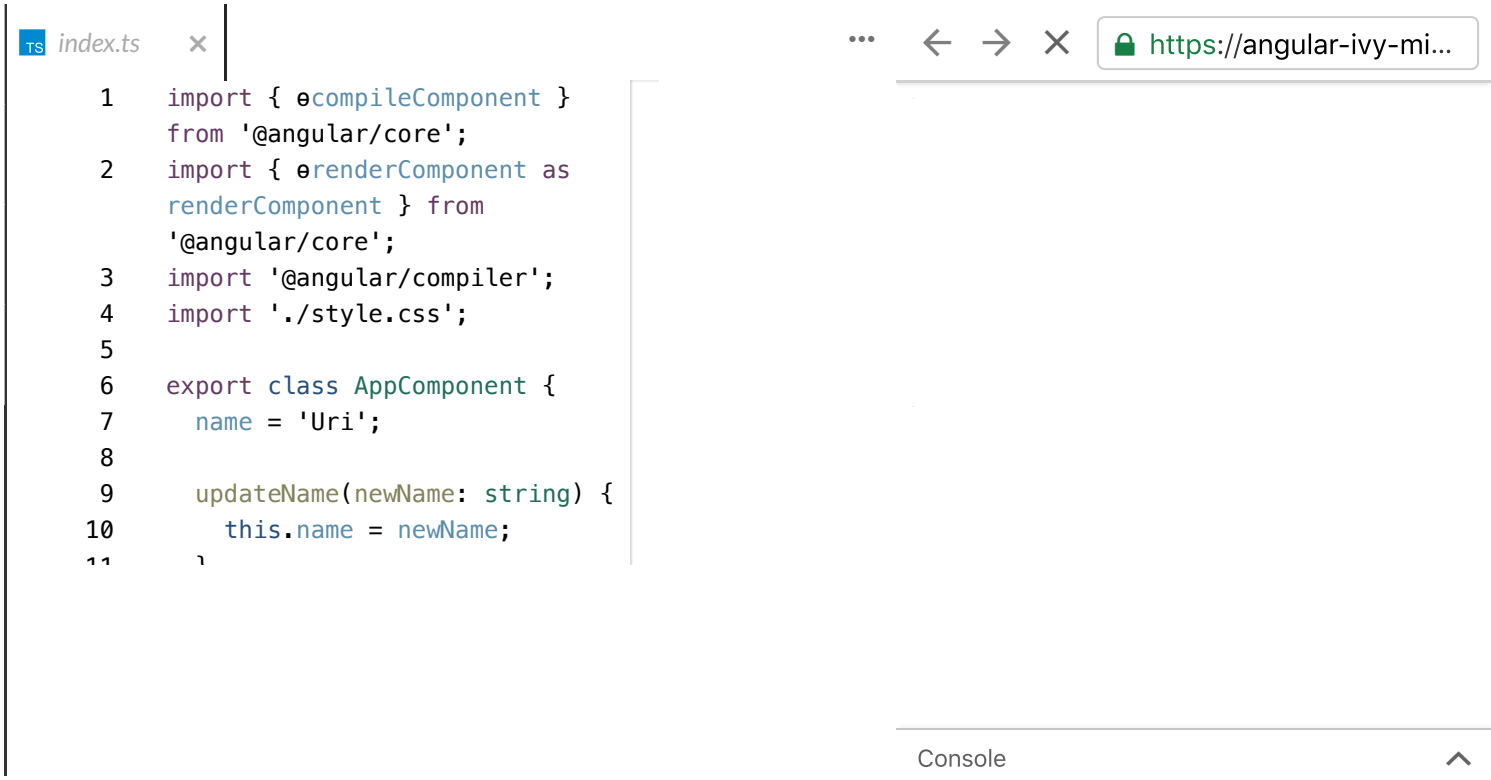
My next thought was: Perhaps this method is exported by Angular, so I can just use it directly in my code on StackBlitz? To my pleasant surprise, it was exported (albeit with the ɵ symbol, which means it is a private API):



According to the doc string, I just had to pass on the component class and the metadata (which is the object with the `selector`, `template`, etc. that is given to `@Component`), and it would do the magic. Excited, I tried that:



Oops! Another error message, but at least a different one, which is a good sign for progress. In addition, the new error message tells us what the problem is: we need to load `@angular/compiler`. I quickly added it to my project and imported it, which seemed to do the trick!



Finally, it worked!

Hooray! The app renders to the screen! But...

Change Detection, Anybody?

After I managed to render my component to the screen, I wanted to check if it was behaving correctly. So I tried to change the text in the input box, hoping that the title would update accordingly. It didn't :/

Putting a `console.log()` inside the `updateName()` confirmed my suspicion—the event listener I defined in my template was indeed firing and calling this method was whenever I typed something into the input box. But, Angular wouldn't pick the changes and update the name in the title.

Why? My best guess was that I didn't use Zone.js, which is what normally triggers change detection in Angular. For some reason, when I tried the same with Angular 6, Ivy triggered change detection in event listeners even without Zone. But, it seems like this undocumented behavior has changed with Angular 7.

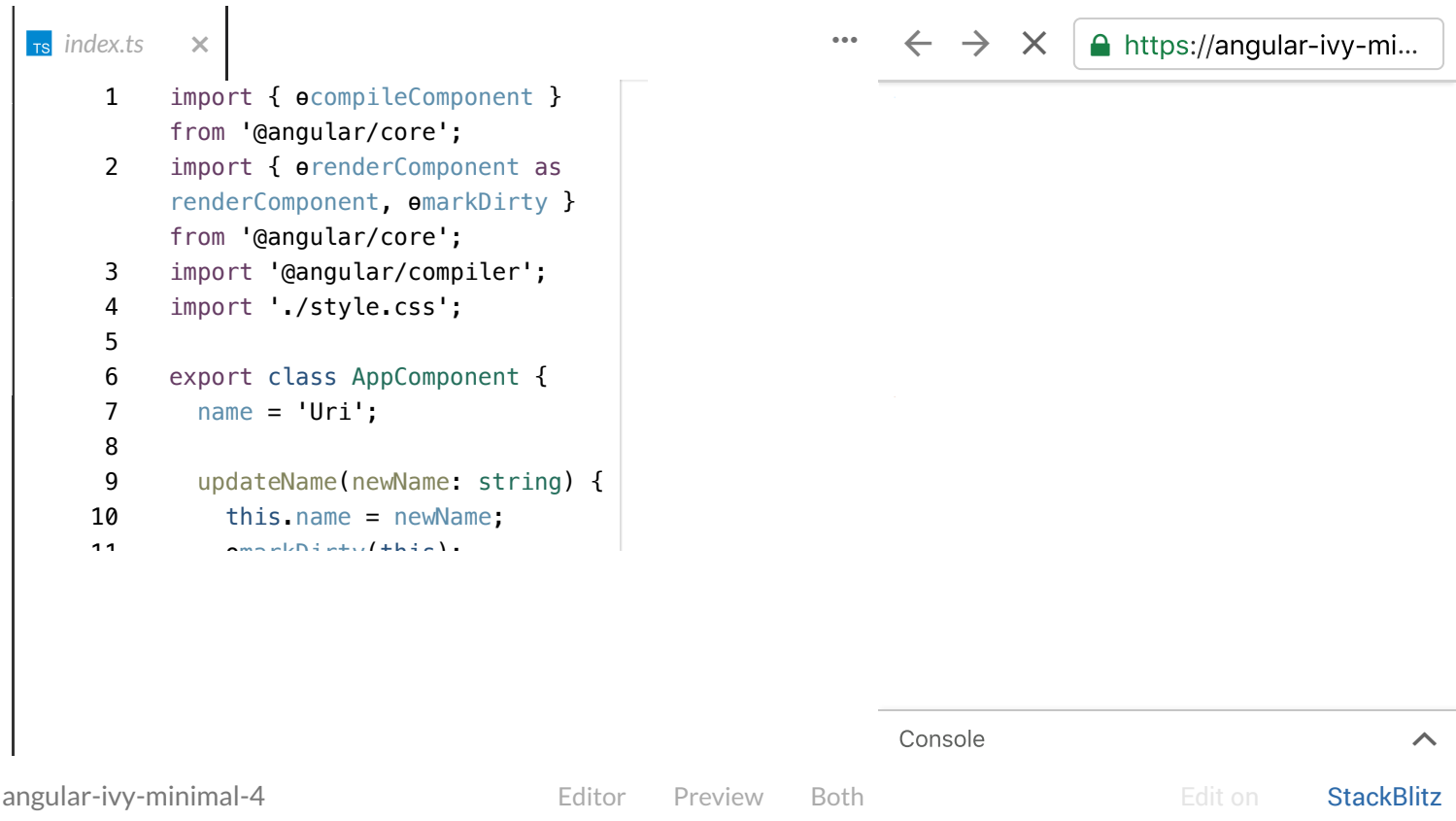
Anyway, I took a nice shortcut here and reached out to Alex Rickabaugh from the Angular team. He's one of the people who actually write Ivy, and even gave a very informative talk about Ivy in AngularConnect this month.

Chatting with Alex proved very helpful. First of all, it helped to demystify why `@Component` was behaving differently in Alexey's code —It turns out that the public Angular builds (the ones available on NPM) disable Ivy compilation for `@Component`, and you have to build Angular with a special flag to enable this behavior (you can find such builds here).

A Dirty Solution

Regarding the Change Detection question, Alex confirmed my guess —apps that bootstrap using `renderComponent()`, like I did above, do not use Zone.js for change detection. If I wanted proper change detection with Zone, I'd have to define an `@NgModule()` for my app and bootstrap it using the `bootstrapModule()` method of `@angular/platform-browser/dynamic` module.

However, Alex also mentioned a simpler alternative: I could manually invoke Angular's change detection by calling a method called `markDirty()`. I tried calling this method in my app's event listener:



Did it do the trick? Well, try for yourself and see!

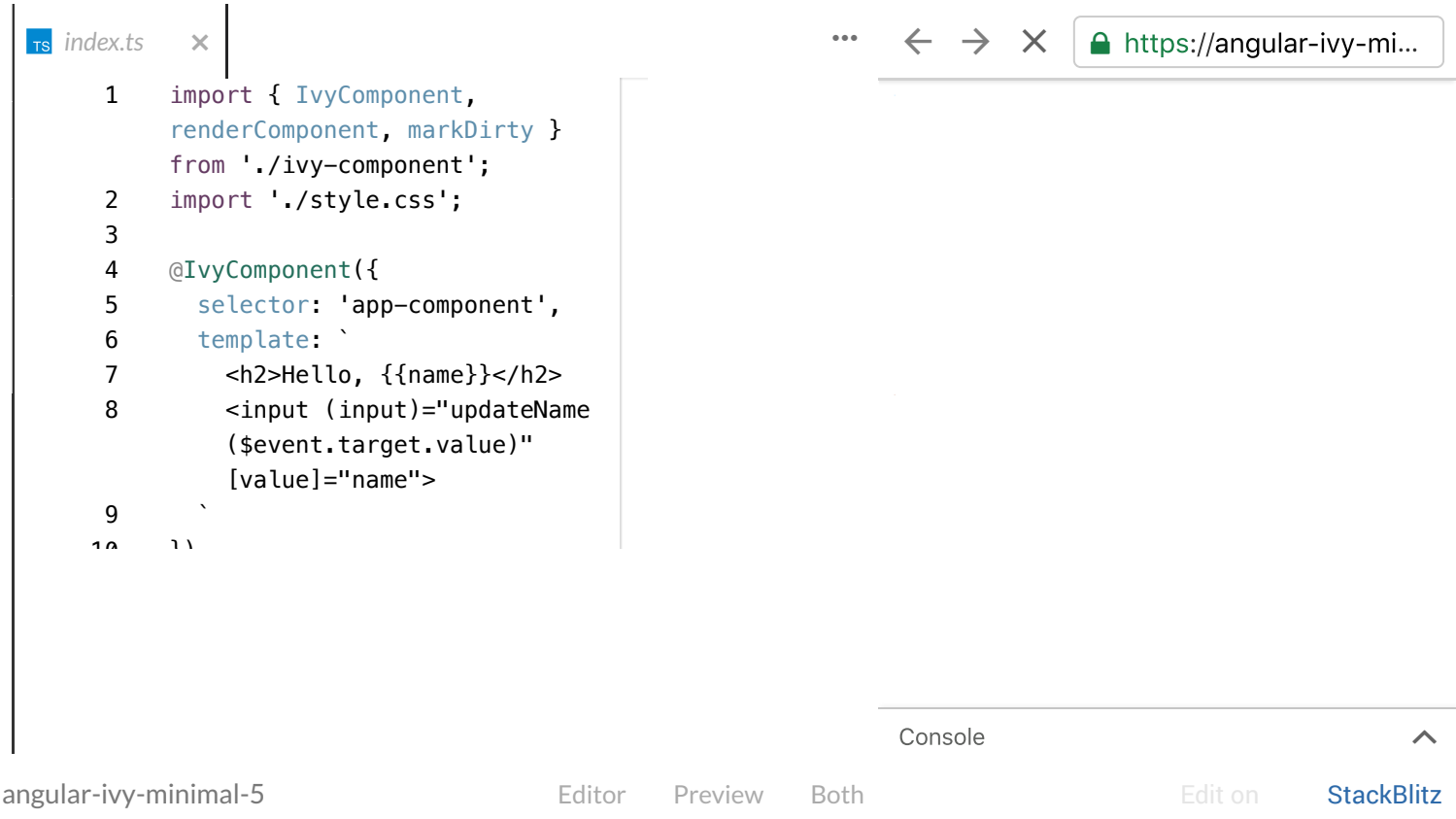
You can use it, too!

I decided to refactor my app, and moving some of the "magic" code that calls the internal method into a different file, called `ivy-component`. It exports an `IvyComponent` directive, which calls the `compileComponent` method on your behalf:

```
import { ɵcompileComponent, Component } from '@angular
export { ɵrenderComponent as renderComponent, ɵmarkDir
import '@angular/compiler';

export function IvyComponent(metadata: Component) {
  return function (clz) {
    ɵcompileComponent(clz, metadata);
```

And this is what the app looks like when using this nice abstraction:



Fork it, experiment with Ivy, and share your findings :)

So here it is—this was my journey to make Ivy work inside StackBlitz, so now Netta and I will be able to use it for our talk next week. But I also hope that this will also whet the appetites of other developers to explore Ivy, especially now that it is as simple as hitting the “Fork” button and tinkering with the code.

• • •

Well, time to go back working on our talk for FrontEnd Con. Which reminds me—if you are in Warsaw next week, you are invited to come and say hi!