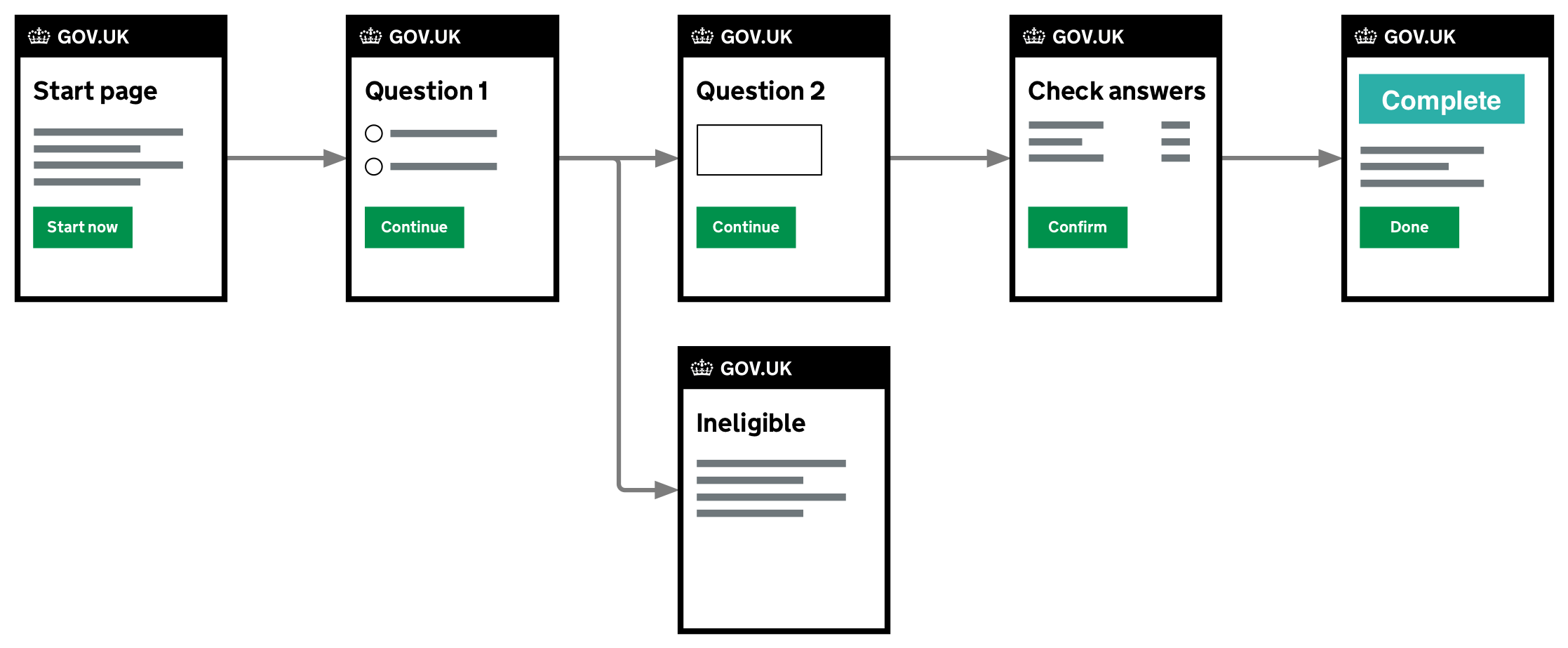
Prototype Kit Tutorial

# We’re going to build a basic juggling licence prototype.

This prototype will ask the user a few questions, tell them if they’re eligible or not based on their answers, and have a basic ‘check your answers’ page that shows their answers.

Our prototype will look a bit like this:



# 

# Lesson 1 - get set up

It's helpful to have the training cheat-sheet open in your browser:

[bit.ly/prototype-training](https://bit.ly/prototype-training)

Get started here: <https://govuk-prototype-kit.herokuapp.com/docs/install>

You should have the kit up and running on your computer before continuing.

Steps to complete:

1. Install the kit and check it works - visit <http://localhost:3000>
2. Open the kit folder in Atom text editor

Overview of kit folders

* **/app** is for your work
  + **views** is HTML
  + **assets** is CSS, JavaScript and images
  + **routes.js** is advanced logic - for example if a user should go to one page or another. We'll cover it later
* **/docs** has examples and templates
* Ignore everything else

# 

# 

# Lesson 2 - basic prototype

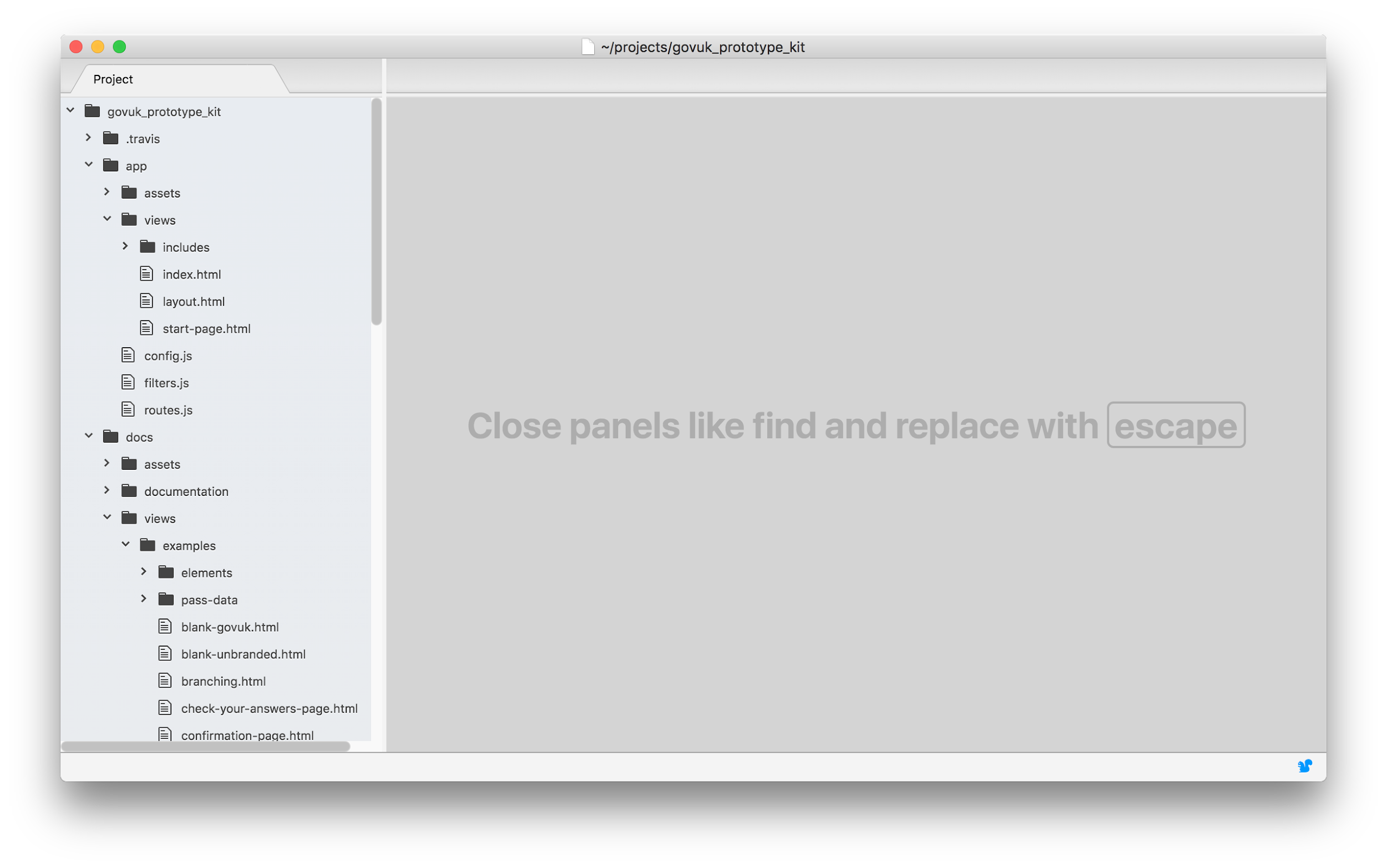
In this lesson we’ll create some pages and link them together so that a user can move from one page to another.

Our prototype pages will go in this order: start page > question page > question page > check your answers page > confirmation page

To create these pages, we’ll make copies of template pages in the kit and rename them. We’ll then link these pages together.

## Creating pages

1. In the file browser on the left, open these folders:
   1. **/app/views** - this is your 'views' folder, where your work lives
   2. **/docs/views/examples** - this is the examples folder, where templates live



1. Copy the start page template to your views folder: in the examples folder, right click **start-page.html**, and **copy**. Right click your **views** folder and **paste**.
2. You can check it works by visiting [**http://localhost:3000/start-page**](http://localhost:3000/start-page)
3. Note that the URL **/start-page** shows the page at **app/views/start-page.html**.   
   For any new page - such as **example-page.html** you can view it by visiting [**http://localhost:3000/example-page**](http://localhost:3000/example-page)
4. In **app/config.js** change **serviceName** to 'Apply for a juggling licence'  
   This is the only thing you need to change in **config.js**, and we put it there so you can change it in one place, and it updates on all the pages in your prototype
5. Create two question pages - the template is **template-question-page-blank.html**. Copy it twice and right-click **rename** to **juggling-balls.html** and **juggling-trick.html**.
6. Edit the h1 in **juggling-balls.html** to be **“How many balls can you juggle?”**
7. Edit the h1 in **juggling-trick.html** to be **“What is your most impressive juggling trick?”**
8. Check these pages work by visiting **localhost:3000/juggling-balls** and **localhost:3000/juggling-trick**
9. Create a check your answers page. The template is **check-your-answers-page.html**. Copy and rename it **check-your-answers.html**.
10. Create a confirmation page from the template

## Link the pages together

If we visit [**http://localhost:3000/start-page**](http://localhost:3000/start-page)and click ‘Start now’ it won’t do anything. We need to link it to the next page.

There are two ways to link pages in HTML: links and forms. We use links when moving between pages that just show content, and forms when submitting user data.

1. Open **start-page.html** (make sure it's your copy in **app**, not the examples folder) and edit the start button link on line **46**. Change **Href** from '#' to **/juggling-balls**: href=”/juggling-balls”
2. Open **juggling-balls.html** and edit the form **action** on line **19**. **Action** should be **/juggling-trick**:action=”/juggling-trick”
3. Do the same for **juggling-trick.html**. The action should point to the check your answers page.

Test your prototype by going to [localhost:3000/start-page](http://localhost:3000/start-page) and clicking the buttons to go through the pages.

# Lesson 3 - Add some questions

In this lesson we’re going to add some radio buttons and inputs to the two question pages.

We’ll do this by grabbing bits of snippet code from the GOV.UK Design System. We’ll copy the code we need into the pages, then update the content to match our questions.

<https://design-system.service.gov.uk>

To start we'll add radio buttons to the first question

1. Open **juggling-balls.html**. We’re going to insert a radio button question in the middle.
2. The page has two placeholder paragraphs. We can delete both of them.
3. Find the **radios** component in the Design System.
4. Copy the **html** of the **Stacked radios** example (there's a copy button on the right).
5. Paste it into **juggling-balls.html** where the old placeholder paragraphs were.
6. It's helpful to make sure the code is indented correctly - select the lines and use **tab** or **shift tab**
7. The html we pasted includes a new **<h1>** tag**.** Change the new h1 text to match the question.
8. Delete the original **<h1>,** and the last radio button - we only need 3.
9. Update the label text to: “**3 or more”**, “**1 or 2”,** “**None - I cannot juggle”**.
10. Change the **value** attribute to the same as the question content above. Make sure to use the exact same capitalisation and spacing.
11. Change **where-do-you-live** to **juggling-balls** on each input  
    Note that the name is the same for all 3 radio options - this represents the question, and the value represents the user's answer

Next we'll add a large text box to the **juggling-trick.html** page.

1. Delete the placeholder **<p>** tags.
2. Copy the html from the **Textarea** example in the Design System.
3. Delete the **label** and the **span** with the hint text **-** we don’t need them, the h1 is the question. (This isn't technically accessible but for the purposes of getting started with prototyping that's ok)
4. Change the name attribute of the input to **name="juggling-trick"**

Check the question pages in your browser to make sure they look correct.

# Lesson 4 - Working with user data

Open **check-your-answers.html** in Atom. Again, make sure it's your version - the one in **/app/views**, not the template in **docs.**

The default page has room for lots of answers - but we just have two. We’ll delete sections from the page until we’re down to two.

Deleting unneeded sections

1. Delete from **lines 90** to **153** - the entire second group of answers.
2. Save and check it looks ok in the browser.
3. We can now delete the second two answers - **lines 57 - 86**
4. Save and check it looks ok.
5. Since we only have one section, we can delete the h2 - delete **lines 19 - 22**

Let's start by making the answers on the page more relevant. You don't have to use the exact question text, you can turn them into statements if that is shorter and clearer.

1. For the first question, change **name** to **Number of balls you can juggle**
2. For the answer, change **Sarah Philips** to **[answer goes here]**
3. For the second question, change **Date of birth** to **Your most impressive juggling trick**
4. For the second answer, change **5 January 1978** to **[answer goes here]**

The change links won’t currently work - we can change the href for each one to point to the correct pages.

The change links have hidden text for screen reader users - we can change this to match the question text. eg **Change<span class="govuk-visually-hidden"> number of balls you can juggle</span>**

## Showing real data

When users enter data, it is saved using the ‘name’ attribute. You can use that ‘name’ later to show answers.

For example when the user fills in an input like this:

<input name="first-name">

The answer can be shown back to the user like this:

First name: {{ data['first-name'] }}

(The curly braces {{ }} are because we're now using a 'templating language' called [Nunjucks](https://mozilla.github.io/nunjucks/))

Let's show the user's answers in the Check your answers page

1. Locate the answer for question 1 on the Check your answers page
2. Replace the answer with **{{ data['juggling-balls'] }}**
3. This gets the data that was saved with the ‘name’ of ‘juggling-balls’.
4. Repeat for question 2 - its name attribute is **juggling-trick**
5. Check that it works by answering the questions and visiting the check your answers page to see the answers displayed. You can try changing an answer and seeing if it updates.

## Editing previous answers

The user’s data is now showing in the Check your answers page, but if they try to change their answer it would be good if they could see their previous answer.

For text inputs we’ll fill data like we did with ‘check your answers’.

1. Go to your **juggling-trick** page
2. Add **{{ data['juggling-trick'] }}** between the **<textarea>** tags (don't add any spaces or newlines. It doesn't normally matter in HTML but they do show up in <textarea>)

Check this works by filling in an answer, continuing to the next page, going back to the page, and refreshing the browser.

For radios and checkboxes, we use a function to mark which of the radios needs to be ‘checked’.

The checked function looks like this: **{{ checked( "name", "value" ) }}**. We need to add it to each of the radio inputs in our **juggling-balls** page.

Insert the **checked** function above inside the first radio input in question 1, like this:

<input id="radio-1" type="radio" name="juggling-balls" value="3 or more” **{{ checked("juggling-balls", "3 or more") }}** >

Note the value in the input and the checked function need to be identical, including capitalisation and spacing.

Repeat for the other two radios, updating the value in the checked function.

Check this works by filling in an answer, continuing to the next page, and then going back to the page. Check that changing an answer also saves.

## Clearing data

If you do research with real user data, it’s important that you clear any data stored by the prototype.

**Always run usability sessions in incognito windows** (In Chrome: **File > New Incognito Window**)**.** At the end of the session, close all incognito windows.

To manually clear data, click on the **clear data** link at the bottom of any page of the prototype.

1. Test clearing data by clicking on the **clear data** link at the bottom of the page.
2. Return to a page that previously had data and refresh - is the data gone?

# Lesson 5 - Branching (showing pages depending on user input)

Our first question asks the user how many balls they can juggle. We’re going to send them to an ‘ineligible’ page if they can only juggle 2 or less.

To do this, we’re going to need an ‘ineligible.html’ page to send them to, and some logic to decide when to send them there.

The logic takes the answer the user has given to the first question, and depending on what the answer was, either show them the next question or send them to the ineligible page.

## Create the ineligible page

1. Make an **ineligible.html** page by copying **template-content-page.html**
2. Update the content to tell the user why they’re ineligible.
3. Update the back link to point at question 1
4. Check it works by visiting <https://localhost:3000/ineligible>

## Route users to the ineligible page

We’re going to write some logic to look at the user’s answer to question 1. If the user can juggle 3 balls or more (the first answer), we’ll show them question 2. If they answer with anything else, we’ll send them to the ineligible page.

Currently, the juggling-balls page sends the directly to question 2. We’re going to instead send them to a new special url where we can run some code to check their answer.

1. In **juggling-balls.html** change the form action (line 16) to **/juggling-balls-answer**
2. Open **/app/routes.js**
3. Insert new javascript into **line 5**, before **module.exports = router**

router.post('/juggling-balls-answer', function (req, res) {

// Make a variable and give it the value from 'juggling-balls'

var jugglingBalls = req.session.data['juggling-balls']

// Check whether the variable matches a condition

if (jugglingBalls == "3 or more"){

// Send user to next page

res.redirect('/juggling-trick')

}

else {

// Send user to ineligible page

res.redirect('/ineligible')

}

})

1. Check it works - test each of the answers.

### If the kit crashes

1. If you don’t get a page, check in the terminal to see if the kit has crashed. This is a common problem if there’s a typo in the javascript. Helpfully the kit will try to tell you the line number with the issue.
2. Sample terminal output:

/Users/edhorsford/projects/juggling-licence-prototype/app/routes.js:12

});

^

SyntaxError: Unexpected token }

at Object.exports.runInThisContext (vm.js:76:16)

at Module.\_compile (module.js:542:28)

at Object.Module.\_extensions..js (module.js:579:10)

at Module.load (module.js:487:32)

at tryModuleLoad (module.js:446:12)

at Function.Module.\_load (module.js:438:3)

at Module.require (module.js:497:17)

at require (internal/module.js:20:19)

at Object.<anonymous> (/Users/edhorsford/projects/juggling-licence-prototype/server.js:7:14)

at Module.\_compile (module.js:570:32)

[14:11:50] [nodemon] app crashed - waiting for file changes before starting…

The first line of this ends with **...juggling-licence-prototype/app/routes.js:12**

The 12 indicates there’s probably a mistake on line 12 or the line before it. In this case, it’s a missing bracket.

# Lesson 6 - finishing touches

We’ve now got a fully working prototype! We’ve got a prototype that looks like GOV.UK, asks some questions, shows answers, and even does some branching.

There’s some final tidying up we can do to it though.

## Back links

You might have noticed that the back links on the pages don’t work. That’s because they need to be updated to link to the previous page.

1. Go through each page and find the html for the back link.
2. Update the **href** to be the url for the page the back link should take the user to.

## Linking to the start page

If we want to visit our prototype, we’ve so far been visiting [**http://localhost:3000/start-page**](http://localhost:3000/start-page). How will someone visiting our prototype find that page though?

We want to link from the index page (<http://localhost:3000>) to the start page.

1. Open **index.html** in your app/views folder.
2. This is the page that is loaded when a user visits <http://localhost:3000>
3. Find a suitable place where you might link to your prototype
4. Decide whether you want to use a link or a button to link to the start page.
5. Find the html you need from the GOV.UK Design System or by copying an existing bit of your prototype.
6. Edit the html to link to **/start-page**

# 

# Lesson 7 - put your prototype online

To put your prototype online, you'll need two accounts: GitHub and Heroku. They're free.

<https://github.com/join>

<https://signup.heroku.com/>

If they ask you about programming language, select Node.js

## GitHub (sharing your code)

Before you can run your prototype online, you need to put the code on GitHub. This is not the only way, but it's relatively easy, and you'll need to use GitHub anyway to share your code with your colleagues.

Some concepts:

**Git** a way to save versions of your code

**GitHub** a site that uses Git to share your code online

**GitHub Desktop** an app that helps you put your code on GitHub

**Repository/repo** a Git project

Download GitHub Desktop from <https://desktop.github.com/>

Run GitHub Desktop, sign in with your GitHub account details.

Click **Add Local Repository** (if it's not on the screen, click the File menu and select it there)

Set the **local path** to your prototype folder

Github Desktop will now show a warning - apologies - this is the correct process!

In the warning, click **create repository.**

On the next screen, titled **Create a repository**, click the button **Create repository.**

In the top right, click **publish repository**

Make sure '**Keep this code private**' is not ticked, as we need others to be able to see and collaborate on it. Click **publish repository.**

Your prototype code is now on GitHub, other people can see and collaborate on your code.

Heroku (running your prototype online)

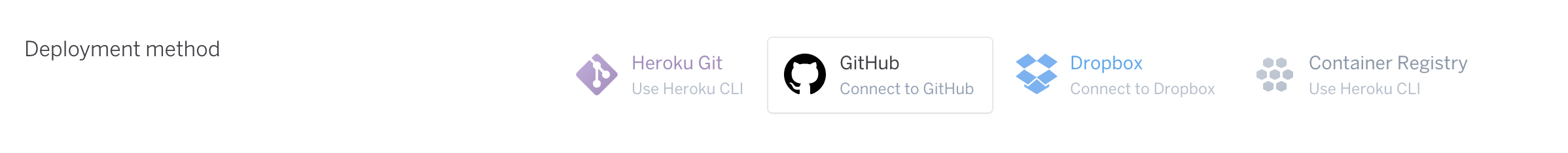
Open [www.heroku.com](http://www.heroku.com) and sign in if you're not already signed in.

In the top right click **New** then **Create new app**

Names in Heroku have to be unique across all the users of Heroku. It can be helpful to add your name or organisation to the start of the name to make it unique. For example **joelanman-juggling-prototype**

Select **Europe** for the region - this is not important but makes your prototype a bit faster.

For **deployment method** choose **GitHub**

****

Scroll down and click **Connect to GitHub**

In the popup, click **Authorize Heroku**

In the **repo-name** field, type all or some of your repo name, as it is on GitHub. Click **search**

Click **connect** on the right of your repo.

Scroll down to the **Manual deploy** section and click **Deploy branch**

One more thing: we need to set a username and password or the prototype won't run online.

At the top click the **Settings** tab

Click **Reveal config vars**

In **KEY** put the word USERNAME

In **VALUE** put a username of your choice, click **Add**

That will be saved and you can add another KEY and VALUE

In **KEY** put the word PASSWORD

In **VALUE** put a password of your choice, click **Add**

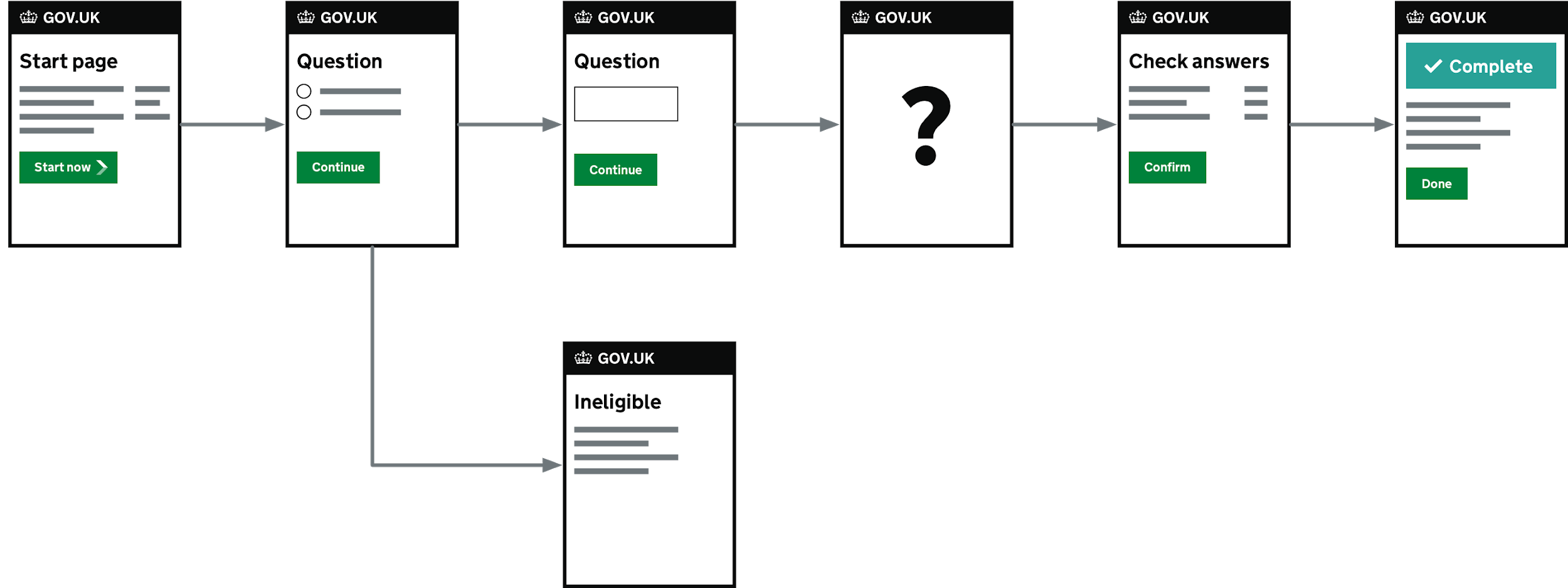
In the top top right, click **Open app** to see your prototype online!

# Lesson 8 - Add a new question

You’ve now got an interactive prototype that can be tested with users. And it’s on the web, so you can share it with your colleagues.

But what if you wanted to add another question?

## Challenge: add a new question



Expand your prototype by adding a new question. You need to decide what kind of question you want to ask. Have a browse through Elements. If you want more of a challenge, choose a question type you’ve not done before.

Steps to do:

* Create a new page using the **template-question-page-blank** template.
* Decide what sort of question you want - and grab the snippets from Elements.
* Add your question to the page and link it to the other pages.
* Add the question and answer to the check-your answers page.

# Lesson 8 - update Heroku

Now that we’ve got a prototype with a new question, let’s update the version hosted on heroku so that others can try it.

Basic steps

* Use Github desktop to commit your new changes
* Push those changes to github
* On heroku.com, tell it to deploy from master

# 

# Cheatsheet

## The Terminal

*Windows users: Terminal is only available in OS X, for Windows use Git Bash, see below.*

**Opening terminal (mac):** open spotlight and type 'terminal'.

The important thing to remember about the terminal is that you're working in one directory (folder) at any one time.

**Commands**

|  |  |
| --- | --- |
| **cd**  *Examples*  **cd public**  **cd ..**  **cd ~** | Change Directory  change to the directory called 'public'  go to the parent directory  go to your home directory |
| **ls** | List everything in the current directory (files and subdirectories) |
| **pwd** | Print Working Directory - show the current path |
| **npm start** | Start the prototype kit (you need to be in your prototype folder) |

Press **up** and **down** on the keyboard to go through previous commands.

Press **ctrl c** to cancel a running app.

Press **tab** to autocomplete a file or folder name. You may need to press it a second time if there are no unambiguous matches.

## Windows users

Get Git for Windows:

<https://git-scm.com/download/win>

This also allows you to install Git Bash, which you can use instead of Terminal. The commands are all the same as Terminal.