30 Day Development Plan

Create a day-by-day gameplan for building your next web application

By Jonathan Calhoun

30 Day Development Plan

Create a day-by-day gameplan for building your next web application

Jonathan Calhoun

Contents

Introduction

As much as we all wish it was, Web Development isn't a well defined process. There isn't a concrete set of steps that you must follow to build every web application, and you will often find that every developer approaches a project in a different manner.

This guide isn't meant to be a definitive guide on how to build web apps, but is instead mean to be a guide that will help you avoid some of the common traps and pitfalls that myself and others have fallen for in the past.

We will cover what parts of your web application will become essentially boilerplate template over time and can be duplicated between projects, which pieces should be given more thought upfront, and how to get a realistic scope of the project you are about to undertake.

Hint: You don't need to build an auth system from scratch for most web apps

If you ever find yourself disagreeing with anything here, that is okay! You are welcome to adapt the plan presented in this guide to suit your needs, and in fact I encourage it. This is simply meant to provide a starting point that you can work from.

Note: We don't follow these steps in the course because we are learning about each tool and I don't find this process as useful when learning about everything for the first time, but I do find it useful when you have a rough understanding of all the pieces and are looking to develop a new web app.

0.1 Why 30 days?

In reality, 30 days is an arbitrary number. You could take less or more time to build your web application depending on how many pages you have in the first version, how complicated it is, and a hundred other factors.

The real goal of this guide is to help you break down the process into small enough steps that you can complete in a reasonable amount of time and will help you find any flaws with your design before you get too vested into building.

That said, I still think most MVPs (Minimum Viable Product) can be built in 30 days or less, and if it is going to take you more than 30 days chances are you are building more than you should in your first version of a web application. In that case, I would encourage you to try to trim up the web application to just the bare essentials so you can complete it quicker and start getting user feedback before adding additional features.

Chapter 1

Week One: Define the user experience

The first few days of building a web application can be counter-intuitive, because I don't actually recommend writing any code. Instead, I suggest spending a little time doing what I call, "defining the user experience."

At a high level this is very simple. We are going to mock out the user experience from their perspective when using your application. When you are just getting started I suggest actually drawing a wireframe of every page, but over time you will get the hang of things and only need to wireframe some pages.

This is much easier to understand with an example, so let's walk through the first few pages of the LensLocked app we build in the course.

1.1 Example: Defining the user experience for LensLocked

To start we need to decide what user experience we are going to define. For example, one flow we might want to consider is what happens when a photographer visits our site, signs up, and then creates a gallery. Another would be the experience of someone visiting our site just to view a gallery.

In this example we are going to follow the path a photographer might take when visiting our site, signing up, and creating their first gallery. In doing this, we would likely draw quick wireframes for the following pages:

1. **Home page/landing page** - this is the first page users see when they visit lenslocked.com. It has links to different pages, but most importantly it has a link to sign up for an account.

2. **The sign up page** - this is where a photographer fills out a form and submits it to create a new user account.

3. **The user's gallery index** - this is where a user is redirected after creating an account. It will eventually show a user all of their galleries, but when the user has no galleries it will simply show a "Create a new gallery" button that the user will click to create their first gallery.

4. **The new gallery page** - the original version of this will just be a form that accepts a gallery name. When it is submitted, the gallery will be created and then the user will be redirected to the edit gallery page for the new gallery.

5. **The edit gallery page** - On the original version of this page a user can upload new images, delete images from the gallery, update the gallery title, and delete a gallery. Eventually we might add more, like publishing/unpublishing a gallery or importing via dropbox.

6. And the list continues...

Even a relatively simple application can quickly end up with quite a few pages. Each might not be very complicated, but it is important to get a sense for how many pages you will need to create because this will help you scope out the project.

As you learn to build web apps you will slowly get a sense for how long each specific page will take you to build, but early on it can often be hard to discern between which pages will be easy and which will take more effort. For example, at first glance the sign up page looks simple, but once you toss in all of the authentication code - hashing a password, creating sessions, etc - it can quickly spiral into a much larger project than you expect.

Because of this, I generally suggest trying to keep the first version of your web application to as few pages as possible. Less than 10 is ideal, but you likely don't want to go any higher than 20 different pages.

If you find yourself with many more pages than this, take a moment to consider which pages you *absolutely* need in order to demo your web application. And remember, your demo doesn't have to be perfect or use every feature. You can fake some of the data or manually enter it into the database. What is important is that you have something to show off your idea and get feedback from users.

1.2 Schedule

1.2.1 Day 1: Create a list of your required pages

Just like we did in the example, make a list of the pages a user will visit in the order they will visit them. Anything that doesn't make that list likely isn't necessary for the first version of your app.

1.2.2 Day 2 - 3: Wireframe each page

Create a wireframe for each page you plan to create. Add input fields for each piece of data you will need on your forms, mock out links to get to the next page you need to visit on every single page. Try to make sure that anything you expect a user to fill in, click, or otherwise interact with is shown in the wireframes.

These also don't need to be digital. I often draft these quickly on a whiteboard and they can look pretty ugly, but they give me a feeling for what all is going to be on the page.

This is more important than it seems because this will help you figure out if you are forgetting something. For example, if you are creating a blogging service and have tags on your "view blog" page but don't have an input field for these it might be a sign that you are missing an important feature of your application.

Creating forms will also make you think about the user experience rather than the final form of the data. This is vital because the best user experience might not map well to the data format you pick at first, but once you know the best user experience you can figure out how to structure the model's data to support that experience.

1.2.3 Day 4: Trim the fat

Spend a day trying to figure out which pages are absolutely necessary vs which aren't. Chances are at least 20% of the pages you are expecting to create aren't absolutely vital, and more often than not you can get rid of more than that.

Remember that web applications aren't like a house or a car. It isn't incredibly expensive or hard to add on later, and you don't have to get it 100% correct before people can use it. A simplified demo that you can get feedback on and then improve is way more valuable than spending months only to find out you built the wrong thing, or users prefer a different workflow than your application provides.

1.2.4 Day 5 - 6: Start to define your resources

We still won't be writing code, but now that you have your wireframes trimmed up spend some time looking at every page and jotting down what resources (models in MVC speak) you will need, what data each will require, and then verify that the pages you wrote have a way for a user to provide this data.

As I said before, you can fake data so it is perfectly acceptable to have models with data you intend to fake, but make note of that in your models so you know about it later.

In the LensLocked.com app we built we might come up with the following resources:

User Account

User accounts map to end users, typically photographers, who come to our site to create galleries.

The user resource will need the following data fields:

- **ID** - Autogenerated
- **Email** - The user's email address, provided via the sign up form
- **PasswordHash** - A hash of the user's password, generated by hashing the password provided on the sign up form
- **Name** - The user's name. We don't *NEED* this, so we can cut it if we want. Provided via the sign up form

Gallery

These are basically containers for images that have a URL that maps to them. Eg mysite.com/galleries/:gallery_id

Data:

- **ID** - Autogenerated
- **UserID** - ID of the user who owns the gallery. Provided automatically by taking the ID of the currently logged in user when the gallery is created
- **Title** - The title of the gallery. Provided via the new gallery form

Image

Images uploaded to a specific gallery.

Data:

- **GalleryID** - We will need some way to retrieve these via a gallery id to get all photos for one gallery. A simple way to do this would be to use folders on the server that each map to a gallery id, then we could just get all images in a folder for a specific gallery. This might need adjusted later, especially if we start supporting Dropbox imports.

- **Path** - We will also need a way to serve these to end users, so we will need a way to map a URL to a specific image so we can link to images in our galleries

Notice that not all of this data is provided by forms, and for some of it I am already thinking about how I might create it. You don't always have to go into this much detail, but at least getting a broad idea of how you intend to access resources and store them will often save you some trouble.

Warning: DO NOT try to get everything perfect up front. I have heard countless horror stories about how developers spend weeks trying to come up with the perfect data schema. The truth is, it doesn't matter! If you get it wrong, you CAN change it and it will take you less time to go that route than if you spend weeks trying to get it right, only to find out you wasted weeks and STILL DIDN'T GET IT RIGHT!

1.2.5 Day 7: Review

At this point you should have a pretty strong understanding of what you are going to be building, but spend some time reviewing everything just to be sure you aren't missing any important details.

If you decide you think you are ready to move on, head on to day 8. Having an extra day of wiggle room won't hurt us in case we run into a roadblock later :)

Chapter 2

Week Two: App structure, authentication, and other boilerplate code

The second week is what I call boilerplate week. Your primary goal here is to simply get a template setup that you can work from, and more often than not this includes roughly the same pieces:

1. An authentication system.
2. An HTML template/design (often using Bootstrap).
3. A general design structure that the rest of your code will follow.

Each of these pieces will vary. For instance, you might decide to use MVC or you might not. Your authentication system might use JWTs, OAuth, or something else entirely. You might decide to use a JavaScript framework and something different from Bootstrap for your frontend.

Those details are completely up to you, and they can change over time, but the important thing right now is to simply pick something and stick with it. You can always change them later on, but I see far too many project fail to get started because people spend too much time trying to design the perfect system.

Don't do that. You will *never* design the perfect system upfront. There is no point in even trying. Just pick something that will work, get it working, then optimize later.

2.1 This may not take a week

Early on when you are just getting started with Web Development I strongly suggest going through each of these pieces in detail. You should write the code for your authentication system, get familiar with Bootstrap (if you are using it), and learn about the design structure (eg MVC) you are using to organize your code.

If you are completely unfamiliar with all of these concepts, it may take much longer than a week to complete this step. But as you grow more familiar with everything, you will eventually get to a point where most of this is boilerplate. Something that can be copy/pasted (or generated) between applications, allowing you to skip this step entirely.

This is what frameworks are often used for - skipping the mundane and repetitive steps of piecing together an overall web app structure, building an authentication system, figuring out what router or database to use, etc.

Even if you don't use a framework, most developers I know don't write this code from scratch with every new web application they build. Instead, they opt to write it the first few times as they become familiar with the code, and then they start using the same boilerplate between web applications so they can essentially skip a week of development and get right to the good stuff.

2.2 What does this look like when complete?

When you have finished this section you should have all of the basic building blocks necessary to build a web application on top of.

That means you should have all of the basic components of your web application picked out. You should know what router you are using. Whether or not you are using MVC, and if you use MVC you should have some example models, views, and controllers (often created to represent users for your authentication system).

If you need an authentication system, then you should have the code that allows users to sign up and log in. You should have a system in place for restricting users form accessing specific pages. You may even need pages where users can update their account details. Exactly what your authentication system needs is up to you to decide, but I find it useful to get this all knocked out upfront so you have a basic idea of how user accounts will work and can leverage them with other pieces of your code.

You should also be able to create new pages relatively quickly with your setup. For example, in the Web Development with Go course we eventually get to a point where creating a new resource is a fairly standard process of creating a model, a service to interact with that model, a controller, and some views. This isn't highly automated, but it doesn't require a lot of imagination on our part because we have a process in place.

I also find it useful to have a rough HTML template at this point. In the course we use a pretty bland default bootstrap template, but you could go as far as creating a custom design or purchasing one from a marketplace as well. This isn't as important as the other pieces, but it is fairly common to want a nice looking site and this isn't a bad time to set one up.

2.3 Schedule

The schedule for week two will vary more than most because it will heavily depend upon whether or not you are using code you already wrote, whether you are trying something new (eg a new JavaScript frontend), and what all is required. For example, many applications don't even require an authentication system in their first MVP, or opt to only provide a single login page via Facebook OAuth and as a result might not need to spend as much time on the user model, hashing passwords etc.

Despite the variance, I have attempted to create a rough outline below.

2.3.1 Day 8: Pick your tools and experiment

The first thing you need to do is pick the tools you are going to use and get used to them. This might involve experimenting with them a bit, or simply just deciding which tools you intend to use.

In the last week we outlined our application, so at this point we should have a fairly decent understanding of what our requirements are but remember that what we can pick here can change over time if necessary.

In the course we don't explicitly do this, but the tools we ended up using were:

- Go's HTML templates package for rendering pages
- Bootstrap to speed up our HTML design
- PostgreSQL & GORM for database interactions
- gorilla/mux for routing
- bcrypt for hashing passwords
- Sessions with remember_tokens for authentication and remembering users

There were likely more details than this, but the main idea here is to try to think about all the tools you expect to use and how they will interact with one another.

2.3.2 Day 9: Pick a design structure

For me this often means deciding on MVC or something else, mocking up what it would look like with my tools, and making sure I'm not missing any details about how my code layout that might be problematic.

If I am being honest, determining when you are missing a detail and what might be problematic is hard at first, but mocking up what you expect your code to look like here using real code is a great step in the right direction and will often expose flaws you might have missed before writing any code.

For example, if you opt to use MVC you might mock up a model, view, and controller and some of the interactions between them for a fictional resources. You don't have to implement each of these, but you should be thinking at a high level like:

> "Okay, the web request is sent to a controller, which then processes data and calls `model.Create` to create the resource. This handles all verification logic. The controller then decides if it needs to render an error or not - if there is an error it renders using `formView.WithError(err)`, otherwise it redirects to the resource just created. This means our views will need a `Render` method and a `WithError` method that does something similar to Render but does so with an error as well."

We might write some of this code out with stubbed methods (I suggest doing this with your first few web apps), but the more important part is that we are thinking about how each piece interacts at a high level and starting to draw lines about responsibilities of each package.

2.3.3 Day 10: Create your templates

Work on creating some HTML (or JS if using a framework on the frontend) templates that you can leverage moving forward.

In the book we do this by creating a Bootstrap layout in the form of a `gohtml` template file, and we set up our navbar along with a few other common pieces our app will need.

2.3.4 Day 11 - 14: Create and test an authentication system

This is one of the few pieces in this guide that won't be broken down day by day, but the basic idea here is to build your authentication system out and test it. This means creating the user model, controllers, and views. Creating any sign up or login pages you will need, and writing some middleware (or something similar) to restrict users from pages they don't have permission to access.

This is all covered in the course, so if you are getting stuck here feel free to reference the book or screencasts.

Chapter 3

Week Three: Building the application

Assuming you are following along, at this point we will hopefully have around 10 different pages we need to create. If you have more it may take you a little longer than 30 days but that is okay - the point of this guide is to help you break things down into achievable tasks, not to figure out how to build every web app in exactly 30 days.

Week three is going to be all about coding. We should already have all of our boilerplate, know how we want to structure our code, and be ready to write code fairly rapidly. Now it is time to execute.

In order to do this we will start with our wireframes and list of pages we need to build and pick a single page to work on. Once we have a single page, we will:

Create the view for the page

This could be in a JS framework, a server side page rendered by a View type, or anything else. The important part is to create the HTML for the page and get it rendering the way you expect.

Determine what work is necessary to make the page functional

This is the trickiest part, because it could vary drastically in how much work is involved. For example, the sign up page might require us to create a user model, write code to hash a password, parse a form, and several other steps.

For every page you build, you will need to figure out what all these steps are that your server needs to perform before finally rendering the next page in your application.

Work on the list of backend tasks

Take your list from the last step and slowly iterate over it, completing the code for each task you need to complete one at a time.

The best way to do this in my experience is to focus on one area at a time and go from there. For example, we might first focus on saving data to a database before even considering validation logic or hashing a user's password. Once we have our models being saved to the database we can start working on the next part, which might be validating that the email address isn't taken or that a password meets a minimum length.

As much as I wish I could give you more guidance here, it is nearly impossible because each task will vary depending on what you are building. Just try to break it down into small steps that you can complete in a day and test. It doesn't matter if it is something you can actually ship to production, but you should be able to test it out locally and verify that it works before moving on to the next piece.

Repeat

Once you complete all the backend tasks you should be ready to move on to your next page and repeat the whole process.

3.1 Schedule

Every page will very in terms of work, but my general suggestion here is to try to spend no more than 1 day on any single page, and for many pages you will spend much less time than that once you get the hang of things.

If you do end up spending more than a day on a page, stop and ask yourself why you are taking so long. Are you doing too much with that page? Are you stuck? Would it be easier to build this page if you understood how other pages work better?

If it makes sense, take a break from that page and move on to others. The only real downside to this is that you can no longer test the entire user workflow from start to finish, but you can often find an easy way to bypass this in the meantime.

3.1.1 Day 15 - 21: Build the app

Follow the pattern described above, each day picking a single page that you need to complete. If you have more than 7 pages you may need to do a few on some days to keep this schedule, or you might need to extend the schedule a bit.

After completing each page take a minute to review your wireframes and the models you defined earlier. Did you miss anything? Does your model have all the data you expected? Does it make sense to change anything, and if so how does that affect the rest of your wireframes and resources?

At this point in the development cycle you are going to be adapting a lot as you learn. You will learn what works vs what doesn't. You will try the user experience firsthand and see if your assumptions were wrong or not and you might need to tweak your design accordingly. Try not to make any sweeping changes that waste all of your work, but don't be afraid to tweak things as you go.

Note: Don't worry about making everything production ready right now - just focus on getting things working. We will spend some time polishing up the app next week.

Chapter 4

Week Four: Deploy the app!

Chances are week three is going to overflow into this week, so I have intentionally kept this week sparse. The work here should take a few days at most, giving you some time to add features you didn't finish in the last week or otherwise giving you more time to ask for user feedback.

4.1 Schedule

4.1.1 Day 22 - 23: Prepare for production

Spend a day or two polishing up your application, getting it read to ship to production. This might include getting rid of hardcoded variables like passwords, API keys, and other things like that, or it might involve adding in some better error handling and logging.

It could also include beefing up your security by adding CSRF tokens, restricting access to pages based on who is logged in, or even creating a few admin-pages so you can more easily manage the application once it is deployed.

We cover a lot of examples of this is in done in Chapter 16 of the Web Development with Go course, but you might want to add some custom changes to your code like integrating with an error handling service (like Rollbar or Sentry).

Note: If you need one, also create a sample config file (or whatever you *intend to use) for demonstrating how to provide these variables to your application when you deploy it.*

4.1.2 Day 24: Set up your server

Spin up a VM or whatever other server you plan to host on and start setting things up for shipping to production.

For me this typically involves:

1. Installing PostgreSQL
2. Installing Caddy
3. Creating systemd config files for caddy, postgres, and our Go app

For you this might include other services or slight changes. Eg you might opt to use docker and need to setup a few dockerfiles during this step.

4.1.3 Day 25: Create a deploy script and deploy configs

If you need one, copy the sample config file from Day 22/23 and update it to store production data.

After that, create a deploy script that can be used to deploy your application using all the pieces we setup over the last few days. This might include uploading a prebuilt binary, building on the server, or using a dockerfile. The goal is to simply set up a relatively simple deploy process.

This should only take you a day. If you find yourself spending more time here I strongly suggest asking yourself if you are making things more complicated than is necessary. For example, using Kubernetes, Docker, and several other tools for an app with no users yet might not be the best use of your time, and you might be instead better served to deploy the app and get feedback on how to improve it.

In my experience you can support quite a few users without getting too fancy with deploy tools, so even though they are useful, they aren't necessary to get your app out there for the world to see.

4.1.4 Day 26: Test your app and ask friends/users for feedback!

Make sure the app is working, try to test it in various nefarious ways to see if you can break it, and then finally send it over to potential users to ask them to test it out and give feedback.

4.1.5 Day 27+: Improve (or play catch-up)

If you finished everything, spend the next few days finding parts of the app that bug you and improve on them. Or if users give bug reports, spend time fixing those.

If you haven't deployed yet, we have a few extra days in the month to play catchup. Just remember not to try to do too much in your first version!

Chapter 5

FAQ

Below are some common questions that students have had as well as answers to go along with them. If I haven't answered a question you have feel free to email me - jon@calhoun.io

5.1 Does it really take a month to build a web application?

The short answer is no. People build web applications in a day or two at hackathons all the time, and then there are some web applications that take months, or even years to get right. The biggest factors here are (a) how well you know your tools, (b) how many features you include, and (c) how "perfect" it needs to be.

At a hackathon developers will often reuse existing code (aka boilerplate code), skip creating a custom design, cut out a few features (like an authentication system), and make some design decisions that wouldn't work out in the long term. All of these decisions are necessary to build something within a few days, but it often doesn't matter because most of the code will be rewritten if the application is successful.

The second version of the application, the one that has all the features, isn't likely to be built as quickly, but will be built in a much better manner. It will be easier to add new features, it will scale better, and it will be a lasting code base. This all takes time, and will likely not be done in a weekend.

Another key here is that most of the people building web apps in a few days have a lot of experience and are using tools they are familiar with. It is much harder to both learn a new technology and build a web app with it in a few days. This is why I would rarely suggest trying a new language out at a hackathon.

5.2 Who is this guide for?

This guide intentionally assumes that you are probably fairly new to web development and will need some time to brush up on skills, will want to learn how to break an application down and estimate how large it is going to be, and more generally that you just aren't very experienced.

Once you know what you are doing you might decide to reuse an authentication system you already wrote, skip a few steps in this guide, or even completing most of this in much less time. That is great, but you shouldn't expect yourself to work at that same pace when you are just getting started.