Appendix D

Self-Assessment Answers

In this appendix, you can find the answers to the end-of-chapter questions.

Chapter 1: Introduction to Angular and its Concepts

1. What is the concept behind Angular Evergreen?

   Answer: Angular Evergreen is about always keeping the version of Angular up to date with the latest release. Keeping up to date is made feasible by sticking to platform fundamentals and avoiding unnecessary third-party libraries.

2. Using the double-click example for reactive streams, implement the following steps using RxJS: Listen to click events from an HTML target with the `fromEvent` function. Determine whether the mouse was double-clicked within a 250 ms timeframe using the `throttleTime`, `asyncScheduler`, `buffer`, and `filter` operators. If a double-click is detected, display an alert in the browser. *Hint: Use* `https://stackblitz.com` *or implement your code and use* `https://rxjs.dev/` *for help.*

   Answer: Refer to the *Reactive data streams* section and the double-click example that is discussed in the chapter. See the live code example below to better understand how a custom double-click handler could be implemented:

   ```
   https://stackblitz.com/edit/ch1-question2
   ```

   ```
   import { fromEvent, asyncScheduler } from 'rxjs';
   import { buffer, filter, throttleTime } from 'rxjs/operators';
   ```

```
const throttleConfig = {
  leading: false,
  trailing: true
}

// click event stream
const clicks$ = fromEvent(document, 'click');

clicks$.pipe(
  buffer(clicks$.pipe(throttleTime(250, asyncScheduler,
throttleConfig))),
  // if array is equal to 2, double click occured
  filter(clickArray => clickArray.length === 2)
).subscribe(() => window.alert('Are you sure?'));
```

Refer to the documentation at `https://rxjs.dev/api` to learn more about `fromEvent`, `throttleTime`, `asyncScheduler`, `buffer`, and `filter`.

3. What is NgRx, and what role does it play in an Angular application?

   Answer: The NgRx library for Angular leverages the Flux pattern to enable sophisticated state management for your applications. There are some excellent reasons to use NgRx; for example, it's a great fit if you are dealing with 3+ input streams into your application. In such a scenario, the overhead of dealing with so many events makes it worthwhile to introduce a new coding paradigm to your project. However, most applications only have two input streams: REST APIs and user input. To a lesser extent, NgRx may make sense if you are writing offline-first **Progressive Web Apps** (**PWAs**), where you may have to persist complicated state information, or architecting a niche enterprise app with similar needs.

4. What is the difference between a module, a component, and a service in Angular?

   Answer: The most basic unit of an Angular app is a component. A component is the combination of a JavaScript class written in TypeScript and an Angular template written in HTML, CSS, and TypeScript. The class and the template fit together like a jigsaw puzzle through bindings, so that they can communicate with each other. A component is the most used directive in Angular.

   A service is a class that can be injected into other services or components via Angular's **Dependency Injection** (**DI**) mechanism.

A module contains metadata about and groups together components, services, directives, pipes, user controls, or other modules to define an Angular application or a feature module. Every Angular application kicks off with a root module. You can organize your application into feature modules, and through lazy loading, you can defer the loading of feature modules until they're needed.

Chapter 2: Setting Up Your Development Environment

1. What are the motivations for using a CLI tool as opposed to a GUI?

   Answer: CLI commands are repeatable and lend themselves very well to automation through scripting. Remember, anything that can be expressed as a CLI command can also be automated.

2. For your specific operating system, what is the suggested package manager to use?

   Answer: For Windows, you can use Chocolatey or Scoop. For macOS and Linux, you can use Brew.

3. What are some of the benefits of using a package manager?

   Answer: Package managers make it easier to install and maintain the software on your computer. They configure CLI tools correctly, so they can run without adjusting any settings manually. Additionally, package managers can upgrade your tools to their latest version without the user having to go to multiple websites or launch multiple tools to discover whether there are new versions. Package managers can also install and maintain GUI tools.

4. What are the benefits of keeping the development environments of the members of your development team as similar to one another as possible?

   Answer: Having identical development environments reduces the chance of a team member experiencing an error due to their development environment configuration. Eliminating environmental factors makes it easier to identify and resolve software bugs. Effectively resolving bugs is a major source of cost and time savings for teams and companies.

Chapter 3: Creating a Basic Angular App

1. I introduced the concept of a Kanban board. What is it, and what role does a Kanban board play in our software application development?

   Answer: Kanban is an Agile software development methodology to account for changes in proprieties and features over time using a prioritized backlog of items that can be worked on. A Kanban board enables collaboration and effortless information radiation amongst team members by organizing backlog items by their status, like to do, in progress, and completed. Using a Kanban board, team members and managers understand the work that is completed, in progress, or should be worked on next, eliminating confusion.

2. What were the different Angular parts we generated using the Angular CLI tool to build out our Local Weather app after we initially created it, and what function and role do each of them serve?

   Answer: We generated three main objects using the Angular CLI:

   - `npx ng generate component current-weather`: This generates an Angular component with a component class as a TypeScript file, a component template as an HTML file, a scoped style as a CSS file, and a unit test file in the .spec.ts format. We can use a component to render information in the browser.
   - `npx ng generate interface ICurrentWeather`: This generates a TypeScript file with a TypeScript interface exported from it. An interface defines the shape of the data you're working with and passing between components and services, or between client and server.
   - `npx ng g service weather --flat false`: This generates an Angular service that can be injected into a component like CurrentWeatherComponent.

3. What are the different ways of binding data in Angular?

   Answer: There are four main types of binding in Angular. The first three of them are different types of one-way binding, and the last one is two-way binding:

   - **Expression binding**: This binding uses the double brackets syntax {{ }} to bind public properties in your component class to the template, so when the value of the property changes, the template is automatically updated.
   - **Property binding**: This binding uses the square bracket syntax [ ] to bind public properties or inline JavaScript code to `@Input` targets in other components. Components can subscribe to change events to react to changes in bound data.

- **Event binding**: This binding uses the parenthesis syntax ( ) to bind to event handlers implemented inline or in your component class. `@Output` event emitters in components trigger changes that can then be handled by your component.
- **Two-way binding**: This binding uses the banana-in-the-box syntax [( )] to implement two-way binding. This means that whether the property is updated in the class, in the template, or through any dependent components, all values are automatically kept in sync. This is the way AngularJS bindings worked by default, and it had severe performance penalties. In Angular, you must explicitly enable this behavior, and outside of limited scenarios, it should not be commonly used.

4. Why do we need services in Angular?

   Answer: Services allow the implementation of business logic in a decoupled, cohesive, and encapsulated manner. Combined with dependency injection, you can easily follow the DRY principle, and only implement the required business logic once.

5. What is an Observable in RxJS?

   Answer: An Observable is the most basic building block of RxJS and represents an event stream, which emits any data received over time. You can think of it as a continuous stream of data that flows in a pipe. The Observable object by itself is benign. It must be activated by calling `.subscribe` on it, which attaches a listener to the event stream. You can implement an anonymous function within the `subscribe` method to handle events. Handlers get executed whenever a new piece of data is received, and an event is emitted. Observables are similar to event-based programming; however, they are more feature-rich through the implementation of `Subject`, which acts as both a listener and an emitter, with configurable behavior to track the history of events.

6. What is the easiest way to present a clean UI if the data behind your template is falsy?

   Answer: Using the `*ngIf` structural directive you can control whether large parts of your template get rendered or not. By doing this you can keep the code within the `ngIf` simple, because you don't have to implement the safe navigation operator for every property.

Chapter 4: Automated Testing, CI, and Releasing to Production

1. What is the test pyramid?

   Answer: Mike Cohn's Testing Pyramid effectively summarizes the relative number of tests of each kind we should create for our applications, taking into account their speed and cost.

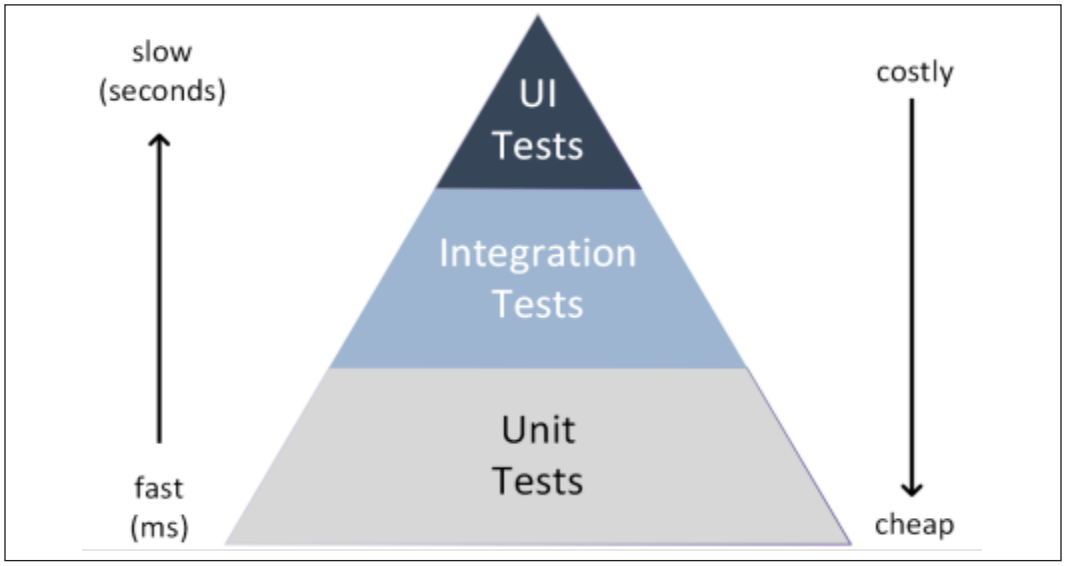

Figure 4.1: Mike Cohn's Testing Pyramid

Automating UI tests is costly, because of their risk of high change, making them brittle. In addition, their execution time is slow, each test taking multiple seconds. This forces developers to wait dozens of minutes before they can iterate over changes. As a result, you should implement multiple orders of magnitude fewer UI tests than your unit tests.

Integration tests exercise the integration of multiple components, so they are at moderate risk of change and they execute faster than unit tests. You should target an order of magnitude less integration tests than your unit tests.

Unit tests are the fastest and cheapest kind of unit tests. They don't depend on any outside data or function. You should implement thousands of unit tests to verify the correctness of your code.

2. What are fixtures and matchers?

   Answer: Fixtures help reduce code repetition in your tests. The following are four common fixtures in Jasmine:

   - `beforeAll()` – runs before all specs in `describe`
   - `afterAll()` – runs after all specs in `describe` per test fixtures
   - `beforeEach()` – runs before each spec in `describe`
   - `afterEach()` – runs after each spec in `describe`

   Matchers assist in asserting the correctness of your tests by comparing expected values to actual values. Below is a list of common Jasmine matchers used in combination with the `expect` assertion:

```
expect(expected).toBe(actual)
                .toEqual(actual)
                .toBeDefined()
                .toBeFalsy()
                .toThrow(exception)
                .nothing()
```

   For the full extent of Jasmine matchers, see `https://jasmine.github.io/api/edge/matchers.html`.

3. What are the differences between a mock, a spy, and a stub?

   Answer: A mock, stub, or spy is used to isolate your unit tests from any external influences, and they do not contain any implementation whatsoever. Mocks are configured in the unit test file to respond to specific function calls with a set of responses that can be made to vary from test to test with ease. A stub is a method with no implementation, either throwing a "not implement" exception or returning a default, often falsy, value. A spy is a wrapper around a mocked property or function with the ability to collect metadata about how outside code interacted with a given property or function, such as the number of times the property or function was called. You can write assertions based on spies to ensure that your code is interacting with external dependencies in the expected manner.

4. What is the benefit of building Angular in prod mode?

   Answer: Angular ships with a robust build tool that can optimize the size of your bundle by removing redundant, unused, and inefficient code from the debug build and pre-compiling sections of code so browsers can interpret it faster. **Angular's ahead-of-time** (**AOT**) compiler, tree-shaking algorithms, and the Ivy rendering engine all play a part in optimizing your app.

So, a 7 MB bundle can become 700 KB and load in sub-second speeds using a fast 3G connection. Prod mode is a critical configuration for the efficient delivery of Angular apps. Do not ship an Angular app without first enabling prod mode.

5. How does GitHub flow work?

   Answer: As GitHub puts it, "GitHub flow is a lightweight, branch-based workflow that supports teams and projects where deployments are made regularly." GitHub flow consists of 6 steps, as shown in the following graphic from GitHub:

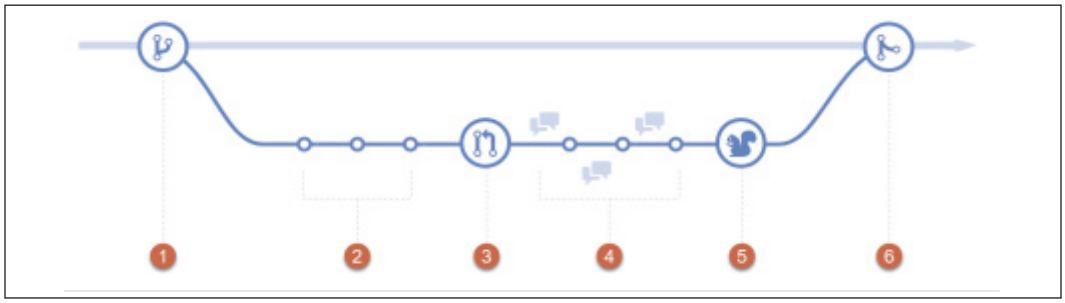

Figure 4.8: GitHub flow diagram

   1. Branch – always add new code for a bug or a feature in a new branch.
   2. Commit – make multiple commits to your branch.
   3. Create a pull request – signal the readiness of your work to your team members and view CI results in a pull request.
   4. Discuss and review – request a review of your code changes, address general or line-level comments, and make necessary modifications.
   5. Deploy – optionally test your code on a test server or in production with the ability to roll back to the master.
   6. Merge – apply your changes to the master branch using GitHub flow. You can ensure that only high-quality code ends up in the master branch. A solid foundation sets other team members up for success when they start making their changes.

6. Why should we protect the master branch?

   Answer: You need to restrict push access to the master branch to enforce GitHub flow and prevent accidental pushes that can rewrite your Git history. In addition, if you implemented a **Continuous Deployment** (**CD**) pipeline, you could accidentally push unvetted code to a dev or prod server.

Chapter 5: Delivering High-Quality User Experiences

1. What are the benefits of using Angular Material?

   Answer: The goal of the Angular Material project is to provide a collection of useful and standard-setting high-quality UI components. The library implements Google's Material Design specification, which is pervasive in Google's mobile apps, web properties, and the Android operating system. Angular Material components are a11y, i18n, mobile, and theming friendly, while being performance conscious and compatible with the latest version of Angular.

2. Which underlying CSS technology does Angular Flex Layout rely on?

   Answer: As the Angular Flex Layout documentation on GitHub aptly puts it "Angular Flex Layout provides a sophisticated layout API using FlexBox CSS and mediaQuery, [prodividing] developers with component layout features using a custom Layout API, mediaQuery observables, and injected DOM flexbox-2016 CSS stylings."

3. Why is it important to test for accessibility?

   Answer: Accessibility, or a11y, is crucial in engaging 100% of your user base. Many users, with varying levels of abilities, rely on automated software like screen readers that can interpret the content of your website in different formats, like audio. It is therefore critical to implement aria labels correctly in your application to ensure full compatibility with such automated tools. In addition, governments and large enterprises must adhere to laws ensuring compatibility with such tools.

   Check out `https://pa11y.org/` for automated verification tools that you can integrate into your CI pipeline to ensure that your application is compatible.

4. Why should you build an interactive prototype?

   Answer: Appearances do matter. Whether you're working as part of a development team or as a freelancer, your colleagues, bosses, or clients will always take a well-put-together presentation more seriously over a rough sketch. But it is also important to quickly communicate ideas without wasting too much time on them.

   A prototyping tool will help you create a better, more professional-looking, mock-up of the app. Whatever tool you choose should also support the UI framework you choose to use, in this case, Material.

If a picture is worth a thousand words, an interactive prototype of your app is worth a thousand lines of code. An interactive mock-up of the app will help you vet ideas before you write a single line of code and save you a lot of code writing.

Chapter 6: Forms, Observables, and Subjects

1. What is the `async` pipe?

   Answer: The `async` pipe automatically subscribes to the current value of an Observable property and makes it available to the template to be used in an imperative manner as a named variable. Additionally, the `async` pipe implicitly manages the subscription to the `Observable` property, so you don't have to worry about unsubscribing from it.

2. Explain how reactive and imperative programming are different and which technique we should prefer.

   Answer: Imperative programming is linear. From the start of your program to the end of it, you must implement every function in sequence. Imperative thinking can only execute on a single core of CPU. As you might be aware, Moore's law doesn't grant us doubling of performance every 12-18 months anymore, so we must rely on a multi-core processor to speed up our code instead. Further, the network stack of our operating systems does not execute on a single thread. They make full use of modern CPUs and networking architecture to leverage multi-threaded workloads to speed up data transfers. This is the reality in which browsers exist — multi-core, multi-threaded CPUs and networking. For our JavaScript code to be efficient, we must take advantage of these features. AJAX was the first implementation of asynchronously loaded content in your HTML to create interactive experiences. However, most asynchronous implementations leverage callbacks, which is an imperative way of thinking about asynchronous problems: i.e. wait until A and B are done before doing C and D. This style of coding doesn't scale when every component of your application becomes asynchronous. Because if C can be executed right as A and B are done independently, we could be wasting time before finally getting to execute D. With reactive programming, we can define composable workloads that express our true intent. i.e. If we can define A → C → D, and B → C → D needs to happen, we don't necessarily care whether A or B gets completed first.

If you multiply this example by 10 or 100 times more variables, you will begin to get an appreciation of how complex Angular and your code sitting on top of it can get. By leveraging reactive coding techniques, you give your code the best chance to execute as quickly as possible.

3. What is the benefit of a `BehaviorSubject`, and what is it used for?

   Answer: The default behavior of Subject is very much like generic pub/sub mechanisms, such as jQuery events. However, in an asynchronous world where components are loaded or unloaded in unpredictable ways, using the default Subject is not very useful.

   There are three advanced variants of subjects:

   - `ReplaySubject` remembers and caches data points that occurred within the data stream so that a subscriber can replay old events at any given time.
   - `BehaviorSubject` remembers only the last data point while continuing to listen for new data points.
   - `AsyncSubject` is for one-time-only events that are not expected to reoccur.

   `ReplaySubject` can have severe memory and performance implications on your application, so it should be used with care.

   Most components you implement will display the latest data received from the server. Through user input or other events, they still need to be able to receive new data so that we can keep them up to date. The `BehaviorSubject` would be the appropriate mechanism to meet these needs.

4. What are memory leaks and why should they be avoided?

   Answer: A memory leak happens when an object is orphaned in memory. This happens because a pointer referencing the object was removed, but the referenced object was left behind. In event-based development, leaks commonly happen through event handlers that are not properly cleared or unsubscribed from. Subscriptions are a convenient way to read a value from a data stream to be used in your application logic. If unmanaged, they can create memory leaks in your application. A leaky application will end up consuming ever-increasing amounts of RAM, eventually leading the browser tab to become unresponsive. This can lead to a negative perception of your app and, even worse, potential data loss, which can frustrate end users.

Chapter 7: Creating a Router-First Line-Of-Business App

1. What is the Pareto principle?

   Answer: The Pareto principle, also known as the 80-20 rule, states that we can accomplish 80% of our goals with 20% of the overall effort.

2. What are the main goals of router-first architecture?

   Answer: Router-first architecture is a way to:

   - Enforce high-level thinking
   - Ensure consensus on features, before you start coding
   - Plan for your code base/team to grow
   - Introduce little engineering overhead

3. What is the difference between the root module and a feature module?

   Answer: The root module defines all the components, directives, pipes, services, and modules that must be present to load your Angular application.

   A feature module encapsulates functionality that can be eagerly or lazily loaded.

4. What are the benefits of lazy loading?

   Answer: Lazy loading enables us to load feature modules into the browser as they are needed. With a lazy loaded architecture, you can ensure that the size of your root module remains consistent, so that your application will load quickly for every type of user you have. As users navigate around your application, they can load parts of it on demand. You can also leverage this feature as an additional layer of security by disallowing users from downloading code related to features they don't need to see. However, remember that real security is always achieved by protecting your server-side endpoints. In addition, you can apply advanced techniques to eagerly load modules before a user is predicted to click on them.

5. Why create a walking-skeleton of your application?

   Answer: It is essential to nail down a walking-skeleton navigation experience of your application early on. Implementing a clickable version of your app will help you gather feedback from users early on. That way, you'll be able to work out fundamental workflow and integration issues quickly. Additionally, you'll be able to establish a concrete representation of the scope of your current development effort.

Developers and stakeholders alike will be able to better visualize how the end product will look. You will also be able to work out any lazy loading issues quickly.

A walking-skeleton also sets the stage for multiple teams to work in tandem. Multiple people can start developing different feature modules or components at the same time, without worrying about how the puzzle pieces are going to come together later on.

Chapter 8: Designing Authentication and Authorization

1. What's in-transit and at-rest security?

   Answer: In-transit security means that your data is secure/encrypted when traveling from point A to point B. This is commonly achieved by using TLS/SSL and HTTPS. HTTPS is the technology that internet commerce relies on to be feasible.

   At-rest security means that your data is encrypted when stored on a file system or a database. In such a configuration, an attacker would need to break into multiple systems to extract data stored in your systems. Most data breaches occur because in-transit or at-rest security has not been properly implemented. In-transit and at-rest security work together to ensure the security of your system.

2. What's the difference between authentication and authorization?

   Answer: Authentication is the act of verifying the identity of a user, and authorization specifies the privileges that a user must have to access a resource. Both processes, auth for short, must seamlessly work in tandem to address the needs of users with varying roles, needs, and job functions.

3. Explain inheritance and polymorphism.

   Answer: Inheritance is an **Object-Oriented Programming** (**OOP**) concept that allows you to adopt the properties and behaviors of a base class. An example of inheritance is a base `Vehicle` class with properties and functions that define fuel level and movement inherited by a `Car` or a `Tank` class. Both `Car` and `Tank` can move forward, backwards, left, or right while consuming fuel.

   Polymorphism is the fact that how a `Car` or a `Tank` moves, and their different rates of fuel consumption, are two quite different implementations.

The value of inheritance and polymorphism is derived when a `Driver` class can drive a generic `Vehicle` without having to change its code. This is similar to how LemonMart can leverage an `AuthService` to implement complex login and logout actions, while the underlying authentication mechanism can be completely different.

4. What is an abstract class?

   Answer: An abstract class is a base class that can be inherited from, but cannot be instantiated as an object. This allows us to encapsulate reusable code, enforceable patterns, and other class-level implementations that can be lent to other classes through inheritance. After all, it doesn't make sense to drive a `Vehicle` that doesn't know how it moves or an `AuthService` that doesn't know how it authenticates.

5. What is an abstract method?

   Answer: An abstract method is defined in an abstract class. An abstract method defines the desired functionality and its signature. It contains no implementation. So, an abstract `turnLeft` function in an abstract `Vehicle` class or an abstract `getCurrentUser` function in an abstract `AuthService` indicates that their inheritors must define the behaviors of those functions.

6. Explain how the AuthService adheres to the Open/Closed principle.

   Answer: Familiarize yourself with SOLID principles. Open/Closed means that your implementation is open to extension but closed to modification. AuthService adheres to this principle because using inheritance, polymorphism, abstract classes, and methods, you can implement any auth provider extending AuthService without changing the implementation of AuthService.

7. How does JWT verify your identity?

   Answer: **JSON Web Token** (**JWT**) implements distributed claims-based authentication that can be digitally signed or integration that is protected and/or encrypted using a **Message Authentication Code** (**MAC**). This means that once a user's identity is authenticated (that is, a password challenge on a login form), they receive an encoded claim ticket or a token, which can then be used to make future requests to the system without having to reauthenticate the identity of the user.

   The server can independently verify the validity of this claim and process the requests without requiring any prior knowledge of having interacted with this user. Thus, we don't have to store session information regarding a user, making our solution stateless and easy to scale.

Each token will expire after a predefined period and due to their distributed nature, they can't be remotely or individually revoked; however, we can bolster real-time security by interjecting custom account and user role status checks to ensure that the authenticated user is authorized to access server-side resources.

8. What is the difference between RxJS's `combineLatest` and `merge` operators?

   Answer: `combineLatest` and `merge` allows us to listen to multiple data streams simultaneously. Every time there's a change in each stream, the pipe we implement gets executed.

   `merge` emits every value from each stream as a single stream. This is useful if you're listening to an event of the same kind from multiple resources, like tweets from multiple people combined in a single list of tweets.

   `combineLatest` emits an array only using the latest values from each stream. This way we can filter out invalid combinations of data, and only trigger our business logic when we encounter a valid combination. This is useful when we need the current user's authentication status and profile information, which come from separate sources, before we can allow the login process to continue.

9. What is a router guard?

   Answer: A router guard can check a given set of conditions, like auth status, user role, or custom logic before allowing navigation to a route. In the case of a feature module, this could mean the difference between downloading a feature module or not.

   As a reminder, auth controls are always enforced on the server side and in your API implementation. Frontend features only serve to improve user experience.

10. What does a service factory allow you to do?

    Answer: A service factory allows you to dynamically inject an implementation of a service at runtime, just like we inject a different auth provider given the `environment.ts` configuration in LemonMart.

Chapter 9: DevOps Using Docker

1. Explain the difference between a Docker image and a Docker container.

   Answer: A Docker image is to a container what a class is to an object. An image for a web server is the declaration and definitions of that server's settings, where the container is the instantiation of your server that is running.

2. What is the purpose of a CD pipeline?

   Answer: **Continuous Deployment** (**CD**) is the idea that code changes that successfully pass through your CI (continuous integration) pipeline can be automatically deployed to a target environment. Although there are examples of continuously deploying to production, most enterprises prefer to target a dev environment with CD.

   A gated approach to deployment is adopted to move the changes through the various stages of dev, test, staging, and finally, production. Most CI systems can facilitate gated deployment with approval workflows.

3. What is the benefit of CD?

   Answer: CD (Continuous deployment) to a target environment, even if it is a dev environment, means that anyone in an organization can continuously inspect and test the work in progress. This is taking the concept of an information radiator as a Kanban board to its functional and living-documentation extreme.

   Extremely sophisticated engineering shops can CD to consumers. However, they also implement A/B testing and sophisticated methods to automatically test the reliability of their deployments.

4. How do we cover the configuration gap?

   Answer: The configuration gap is when a set of configuration works in a development environment, but it fails in a production environment. It happens between the time code is committed and shipped, as shown in the diagram below:

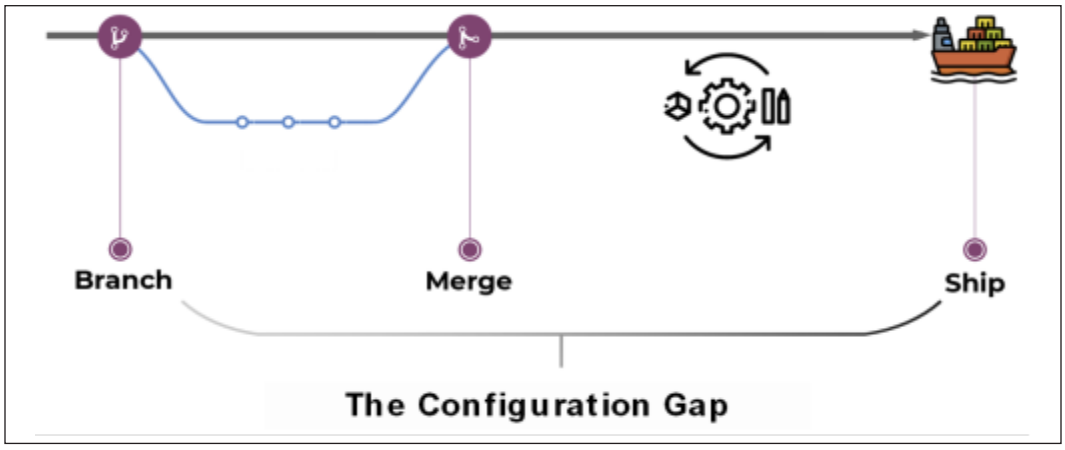

Figure 9.3: Covering the configuration gap

The configuration gap is closed/covered when we leverage **Infrastructure-as-code** (**IaC**) and Docker to create repeatable environments at every stage of development and deployment.

5. What does a `CircleCI` orb do?

   Answer: An orb encapsulates the configuration of prerequisite CLI tools for a CI job.

6. What are the benefits of using a multi-stage `Dockerfile`?

   Answer: By leveraging a multi-stage `Dockerfile`, we can define a customized build environment, and only ship the necessary bytes at the end of the process. For example, we can avoid shipping 250+ MB of development dependencies to our production server, and only deliver a 7 MB container that has a minimal memory footprint.

7. How does a code coverage report help maintain the quality of your app?

   Answer: Code coverage reflects the percentage of code that is covered by unit tests in your application. You can use code coverage as a quality control gate in your GitHub flow to ensure that new code isn't reducing your overall coverage. This is a great way to reinforce the **Test-Driven Development** (**TDD**) mindset on your individual projects or with your team. However, a high code coverage metric alone is not an indicator of high quality. We must test the right things and avoid testing functionality that the framework or library provides for us. TDD is a deep topic. I recommend Kent Beck's *Test-Driven Development: By Example* as a good resource.

Chapter 10: Restful APIs and Full-Stack Implementation

1. What are the main components that make up a great developer experience?

   Answer: The three main components are:

   - Ease of use
   - Happiness
   - Effectiveness

   If your developers will be working on your application for an extended period of time, then it is very important to consider factors beyond compatibility. Your stack, choice of tool, and coding style can greatly impact whether or not your code base is easy to use, keeps your developers happy, and makes them feel like effective contributors to the project.

A well configured stack is key for a great **Developer Experience** (**DX**). It can be the difference between a towering stack of dried out pancakes or a delicious short stack with the right amount of butter and syrup over it.

By introducing too many libraries and dependencies, you can slow down your progress, make your code difficult to maintain and find yourself in a feedback loop of introducing more libraries to resolve the issues of other libraries. The only way to win this game is to simply not play it.

2. What is a `.env` file?

   Answer: A file that stores environment variables that are specific to your environment. `.env` files often include secrets like passwords and API keys, so they should never be committed to your Git repository. An accidental exposure of a key or a password means that you must immediately rotate the old secret with a new one, as your Git history is immutable.

3. What is the purpose of the authenticate middleware?

   Answer: Middleware in Express.js can execute before an API endpoint/ implementation executes. The authenticate function is a middleware that we use to ensure that only authenticated and authorized users can access any given endpoint. Note that this is the only real way to enforce the security of your system, so you must pay great attention to the proper implementation of your API security. You can apply multiple middleware to an endpoint. For example, the rich ecosystem of Express.js includes rate-limiting middleware that can thwart attacks that attempt to guess a user's password, by artificially slowing down server response times.

4. How is Docker Compose different than using a Dockerfile?

   Answer: A `Dockerfile` defines a single executable environment, whereas a Docker Compose file or `docker-compose.yml` can describe an infrastructure with multiple environments using private networks to interact with each other and define storage volumes to persist their data.

5. What is an ODM? How does it differ from an ORM?

   Answer: An **Object Document Mapper** (**ODM**) represents documents in a document-based datastore, like MongoDB, as objects in your application code, giving you a natural way to interact with persisted entities. Mongoose and DocumentTS are examples of ODMs. An **Object Relation Mapper** (**ORM**) does the same thing for relational databases. Entity Framework and NHibernate are examples of ORMs.

6. What are the uses of Swagger?

   Answer: Swagger allows you to design and document your web API. For teams, it can act as a great communication tool between frontend and backend developers, reducing a lot of friction. Defining your API surface early on allows implementation to begin without worrying about late-stage integration challenges.

7. How would you refactor the code for the `/v2/users/{id} PUT` endpoint in `userRouter.ts` so the code is reusable?

   Answer: You can move the logic of updating a user into a new function in `userService.ts`, so it can be reused from multiple endpoints.

   Observe the differences between the new `userService.ts` and `userRouter.ts` files as shown below:

   server/src/services/userService.ts

```
export async function updateUser(
  userData: User,
  userId: string
) {
  const _userId = new ObjectID(userId)
  delete userData._id
  await UserCollection.findOneAndUpdate(
    { _id: _userId },
    {
      $set: userData,
    }
  )
  const user = await UserCollection.findOne({ _id: _userId })
  return user
}
```

   server/src/v2/routes/userRouter.ts

```
router.put(
  '/:userId',
  authenticate({
    requiredRole: Role.Manager,
    permitIfSelf: {
      idGetter: (req: Request) => req.body._id,
      requiredRoleCanOverride: true,
    },
  }),
```

```
  async (req: Request, res: Response) => {
    const userData = req.body as User
    const userId = req.params.userId as string
    const user = await updateUser(userData, userId)

    if (!user) {
      res.status(404).send({ message: 'User not found.' })
    } else {
      res.send(user)
    }
  }
)
```

Chapter 11: Recipes – Reusable Forms, Routing, and Caching

1. What is the difference between a component and a user control?

   Answer: User controls are inherently highly reusable, tightly coupled, and customized components to enable rich user interactions. Rich user interactivity usually requires complicated code that manages DOM interactions. On the other hand, components implement business logic and should be kept as simple as possible, so they are easy to maintain and modify to accommodate ever-changing business requirements. Note that this is a conceptual differentiation. Both concepts are implemented as Angular components. However, a user control usually also implements the `ControlValueAccessor` interface.

2. What is an attribute directive?

   Answer: Attribute directives allow you to define new attributes that you can add to HTML elements or components to add new behavior to them.

3. What is the purpose of the `ControlValueAccessor` interface?

   Answer: The `ControlValueAccessor` interface standardizes change detection in user controls, so that your custom controls will play nicely with forms and the form validation engine.

4. What is serialization, deserialization, and hydration?

   Answer: An object in memory is serialized to textual form so it can be exchanged between components. This can mean transferring data over the wire, storing data in a database, or passing data between components.

Serialized data is deserialized to reconstruct the data as an object. In JavaScript, JSON is a generic object, so deserializing to JSON may not be enough. To reflect the true nature of the data received, we hydrate a class that represents the correct type by instantiating a new object and filling it with deserialized data. In the example of the User object from the book, this allows us to use calculated properties like `fullName` in our application code.

5. What does it mean to patch values on a form?

   Answer: The `patchValue` function on a `FormGroup` allows us to merge the existing values of the form with a set of new properties provided either partially or completely. Patching values makes it efficient to update only a few values, otherwise we would have to rebuild the `FormGroup` object from scratch with every update or set each input's value individually.

6. How do you associate two independent `FormGroup` objects with each other?

   Answer: By leveraging an abstract `BaseForm` class, we can define a common interface for forms to interact with each other. A child form can implement an `@Output() formReady: EventEmitter<AbstractControl>` property and a parent form can listen to this event.

   Once instantiated, the parent can register the child form when it's ready by listening for the `formReady` event with `(formReady)="registerForm('name', $event)"`.

   `registerForm` leverages `FormGroup's setControl` function, which seamlessly integrates the child form as a part of the parent form. In this new form, validation controls continue to work as expected, which is a big win for reducing code complexity.

Chapter 12: Recipes – Master/Detail, Data Grids, and ngrx

1. What is a resolve guard?

   Answer: Resolve guards are used to reduce boilerplate and asynchronous code handling, by synchronously providing loading data before navigating to a component.

2. What are the benefits of router orchestration?

   Answer: Router orchestration is used to orchestrate how components load data or render via route definitions and dynamically defined `routerLink` attributes. Using router orchestration, you can define distinct UI layouts using a combination of components without having to define new components.

3. What is an auxiliary route?

   Answer: Auxiliary routes are routes that are independent of each other where they can render content in named outlets that have been defined in the markup, such as `<router-outlet name="master">` or `<router-outlet name="detail">`. Furthermore, auxiliary routes can have their own parameters, browser history, children, and nested auxiliaries. Auxiliary routers are key in enabling rich router orchestration workflows.

4. How is NgRx different than using RxJS/Subject?

   Answer: The NgRx library brings reactive state management to Angular based on RxJS. State management with NgRx allows developers to write atomic, self-contained, and composable pieces of code, creating actions, reducers, and selectors. This kind of reactive programming allows side-effects in state changes to be isolated. In essence, NgRx is an abstraction layer over RxJS to fit the Flux pattern.

   The Flux pattern requires many parts to express a centralized store of information, whereas an `RxJS/Subject` is only a line of code per object tracked to express a similar intent.

   Deciding whether to use NgRx in your project or not requires a strong grasp of what it entails to implement an NgRx application and how far you can push out-of-the-box RxJS functionality.

   Note that you can limit the implementation of NgRx in feature modules. So, if only a portion of your application is real-time and complicated enough to warrant NgRX, it may make more sense to implement a hybrid approach.

5. What's the value of NgRx Data?

   Answer: If NgRx is a configuration-based framework, NgRx Data is a convention-based sibling of NgRx. NgRx Data automates the creation of stores, effects, actions, reducers, dispatches, and selectors. If most of your application actions are **Create, Read, Update, and Delete** (**CRUD**) operations, then NgRx Data can achieve the same result as NgRx with a lot less code.

   NgRx Data may be a much better introduction to the Flux pattern for you and your team than NgRx itself.

6. In `UserTableComponent` why do we use `readonly isLoadingResults$: BehaviorSubject<Boolean>` over a simple Boolean to drive the loading spinner?

   Answer: To avoid the `ExpressionChangedAfterItHasBeenCheckedError` of course! When using asynchronous data in your templates that update during a life-cycle change event of your component, you will get the error `ExpressionChangedAfterItHasBeenCheckedError`.

You need to use an RxJS/Subject and use the async pipe to bind the value in your template, so no matter when your data arrives the template will react to the new data appropriately and do so without leaking memory.

Chapter 13: Highly-Available Cloud Infrastructure on AWS

1. What are the benefits of right-sizing your infrastructure?

   Answer: Right-sizing your infrastructure has massive cost benefits to your organization, while maintaining high levels of consumer satisfaction. If you provide too few resources, your customers will complain about slow performance. If you over-compensate, then you will needlessly pay for infrastructure that you're not using.

2. What is the benefit of using AWS ECS Fargate over AWS ECS?

   Answer: AWS ECS Fargate abstracts physical servers from your cluster configuration, whereas with AWS ECS you must specify your own EC2 instances. In a niche deployment scenario, you may be required to use AWS ECS, since Fargate is optimized for generalized use cases.

3. Did you remember to turn off your AWS infrastructure to avoid getting billed extra?

   Answer: This is a reminder for you to set your ECS service to require zero instances, and ensure your ALB and EC2 instances are deprovisioned so you don't incur continued charges on your AWS bill.

4. What is blue/green deployment?

   Answer: Blue/green deployment enables you to deploy a new version of your application without downtime. Blue/green deployments make sure that your new deployment is healthy before draining connections from existing servers to new ones. When drained, the old servers are decommissioned, and all users are seamlessly transitioned over to the new servers. If the new deployment is not healthy, the deployment will fail, avoiding an outage. A full-stack stateless architecture is the easiest way to enable blue/green deployments.

Chapter 14: Google Analytics and Advanced Cloud Ops

1. What is the benefit of load testing?

   Answer: Stressing your system using a realistically modelled load test ensures that your infrastructure is right-sized to meet customer demands. This is critical to ensure that your infrastructure spending is kept to a minimum while ensuring that customers are satisfied with the performance of your system.

   As my colleague Brendon Caulkins points out, load testing can also point out code that is poorly performant, and help identify code-level issues that cannot be solved with "more horsepower" and should be looked at by the development team.

   Good performance is good UX.

2. What are some considerations for reliable cloud scaling?
   - Real-time monitoring of infrastructure metrics
   - Dynamic scaling of infrastructure
   - Well calibrated scale-in/scale-out metrics
   - Blue/green deployments

3. What is the value of measuring user behavior?

   Answer: Tracking user behavior allows you to measure user engagement. In web applications, user engagement can validate or invalidate your entire business model. So, you must track user behavior in order to run a successful and efficient business.