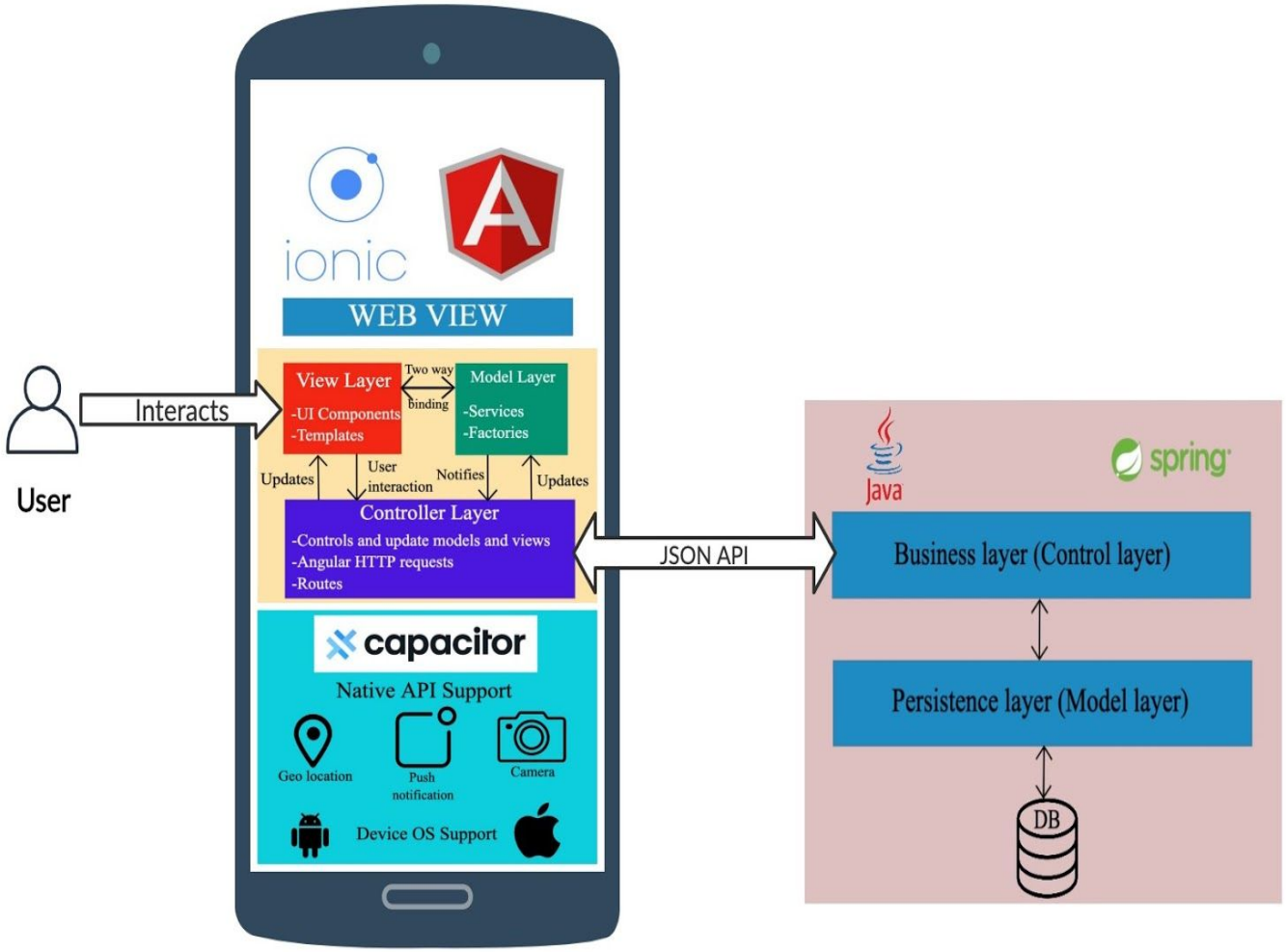
SYSTEM ARCHITECTURE OVERVIEW OF DIGITACT APPLICATION



Additional Explanation:

For the backend JAVA platform is used with spring boot. The layers of spring boot are explained below.

Front end Layer:

It is the View layer that handles the HTTP requests, and communicates with the Business layer through JSON API. Ionic with Angular is chosen as a framework for developing hybrid applications.

How is Ionic structured?

In general, there are two approaches that current hybrid app frameworks follow. Either, they take your source code and compile it into real native code (as done in React Native, Flutter or similar). Or, they simply directly render the HTML/CSS/JS you wrote in a WebView. The latter is the way Ionic works.

“Ionic” is a term that is used by developers to either reference only a set of UI components, or to reference the whole framework. To stop this confusion, the following (partially made up) terms will be used:

- Ionic UI: The UI Components provided by Ionic (the term “Ionic UI” actually doesn’t really exist)
- Capacitor: The bridge between JavaScript and the native SDKs. So basically “the compiler that takes JavaScript and generates a fully working app”.
- IonicFramework: Both the UI components and Capacitor

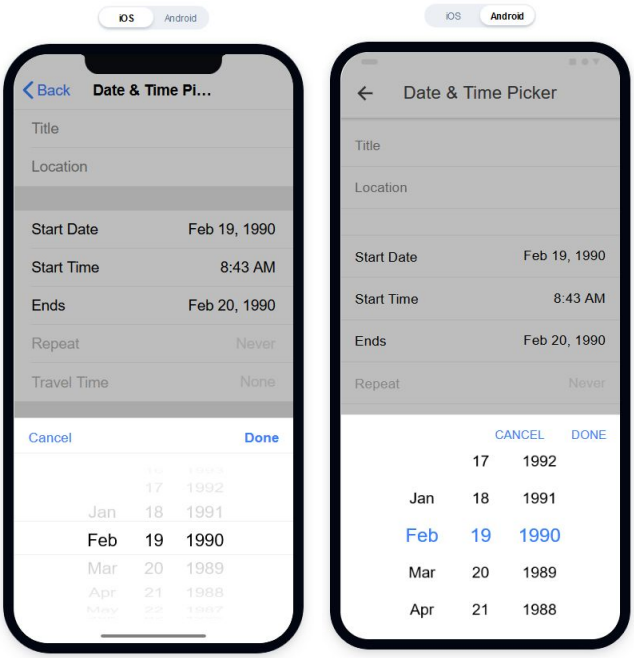
Ionic UI:

Ionic provides a huge set of UI Components allowing you to create an application that looks just like a native iOS or Android app. You only need to write your code once and Ionic will apply different styles to it, depending which OS (iOS/Android) it is currently running on. When developing in your Browser (for instance using the Chrome Devtools), you only need to change your User Agent and reload the page. Ionic will automatically switch from Android to iOS mode. So, you can easily preview your app in your own browser and don’t need to install any native compiler or similar. Developers just need to keep in mind that it is not 100% accurate (e.g. the native APIs are missing, there is no “safe area” (the part of the screen on the latest iPhone where you shouldn’t put UI elements) or some fonts/styles might look different).

Examples:

You can find examples in the Ionic Documentation.

In the following, you can find one example from the docs. On the left, when running the app on iOS. On the right, when running it on Android.



Angular

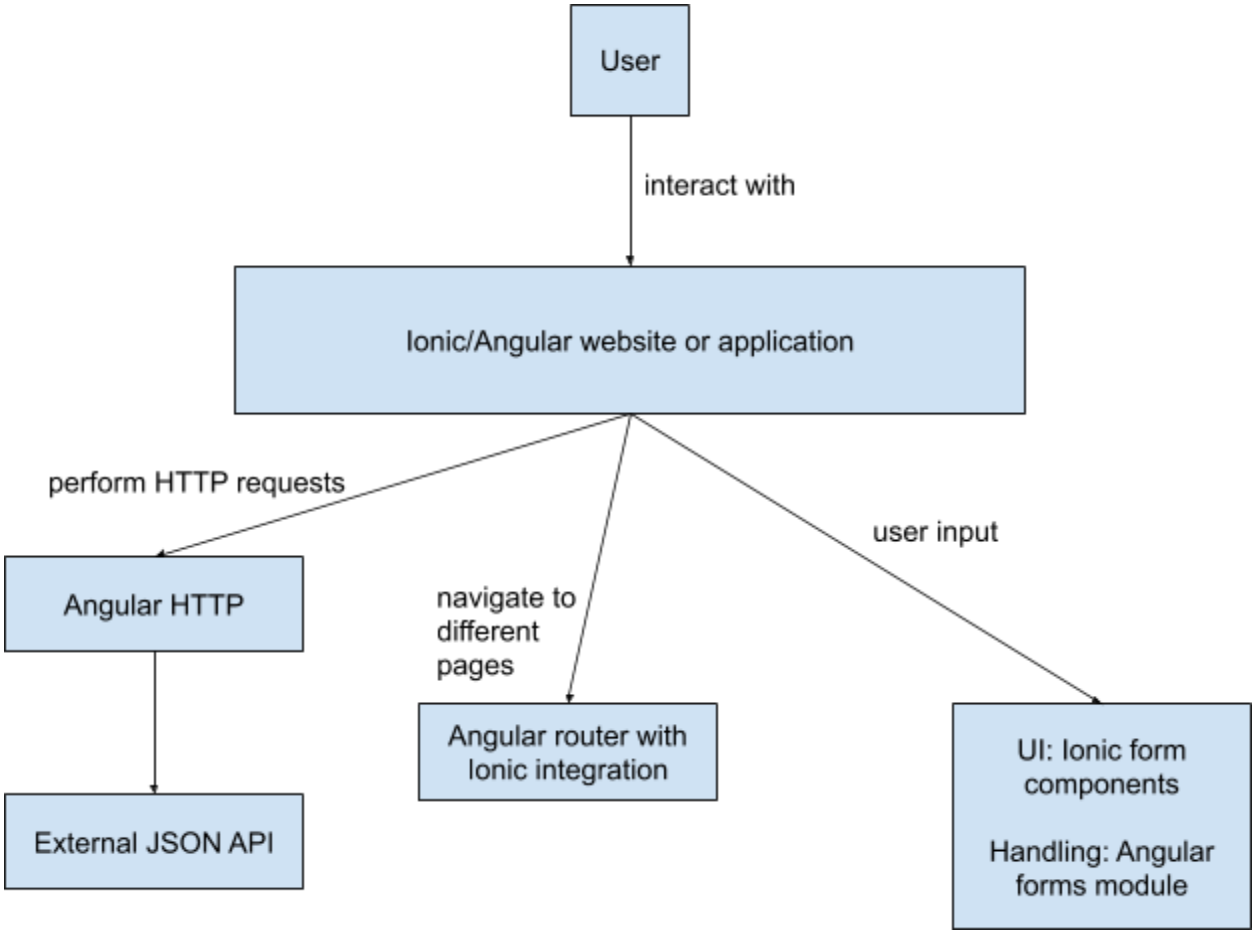
You can use the Ionic UI components in any JavaScript project. However, in modern web development, you should choose a framework/library like Angular, React or Vue to simplify your development process. We have decided to use Angular. Ionic traditionally only supported Angular, and later (as of the release of version 4) opened up to React and vanilla JavaScript (this term means "without any particular framework/library"). So, the integration of Angular can be considered the most "mature" one.

Structure of the website/app:

For most applications, the most important functionalities are:

- Making HTTP calls
- Routing (handle the navigation between different pages. So: “Which piece of code should I execute when the user navigates to page X?”)
- Handling of forms (and other kinds of user input)

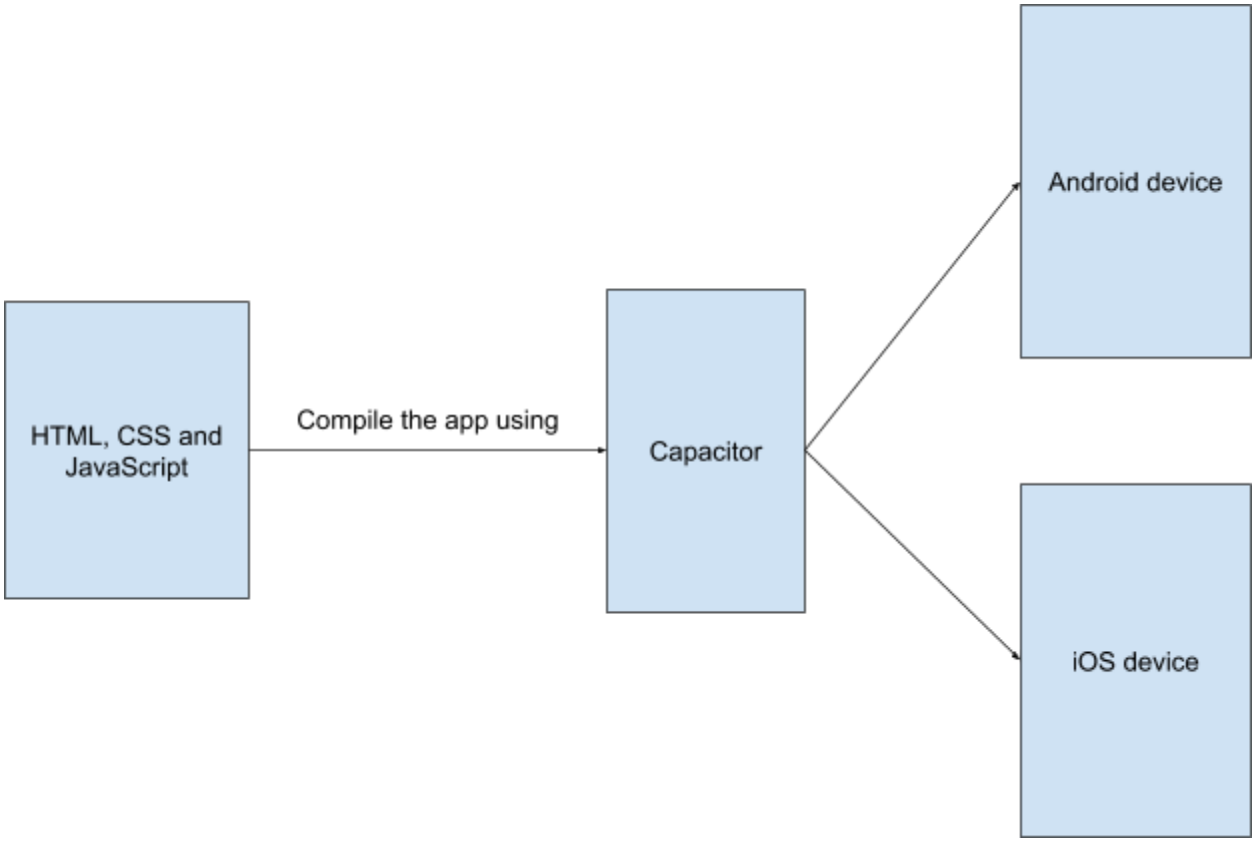
Luckily, Angular/Ionic already provide all of the above (and much more) out-of-the-box.



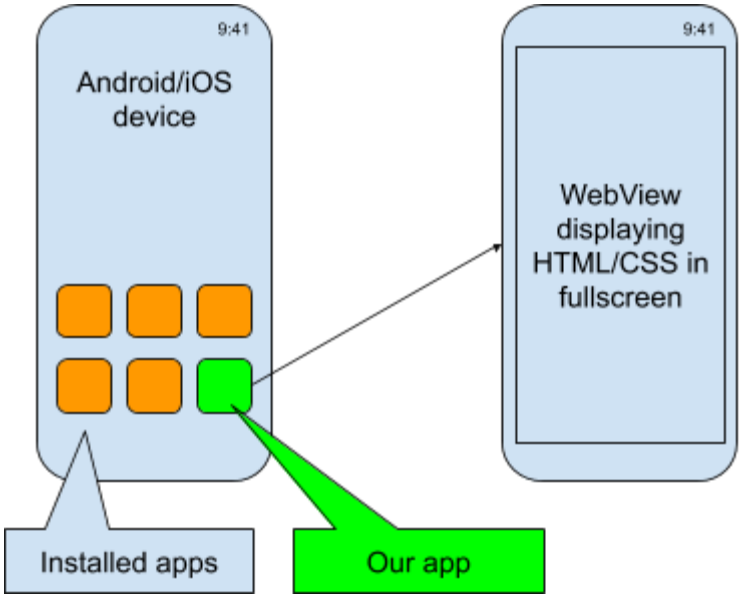
This structure allows most of the development work to be actually done in the browser, since only web technologies are used here.

Capacitor

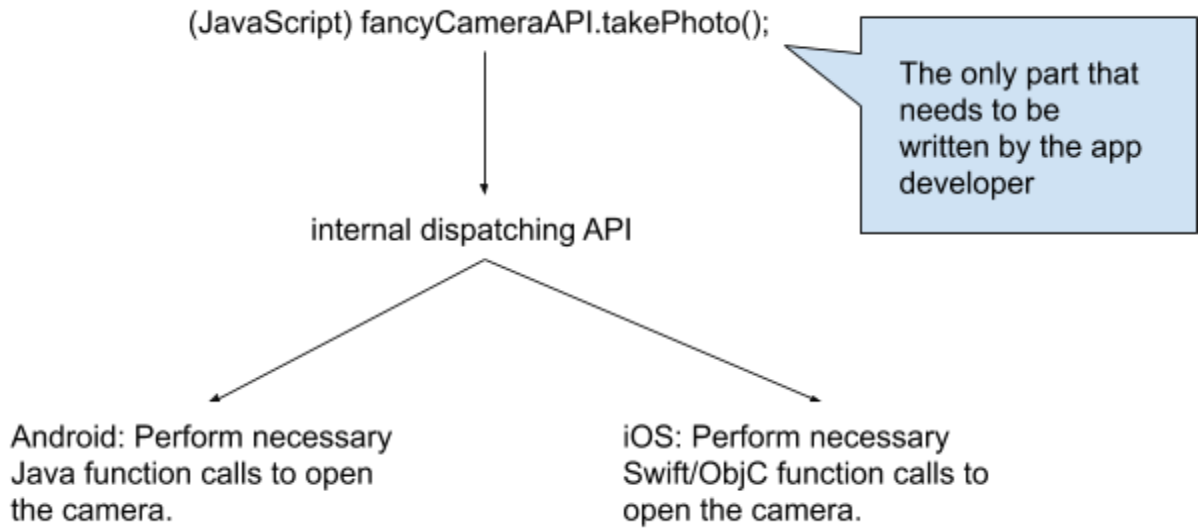
To actually generate the installable Android (.apk) and iOS (.ipa) app, Capacitor is used.



In really simplified words, Capacitor displays a full screen web view and loads the JavaScript, HTML and CSS into it.



Sometimes, you need to use a native API, e.g. to open the camera. This is done using a JavaScript-to-Native bridge provided by Capacitor. It might look somewhat like this:



One issue that might arise here: If there is no such "fancyCameraAPI" available, this needs to be implemented separately, using native Java and/or Swift code. Implementing native functionality like this makes it harder to preview the app in the browser. Because of this, either a browser fallback should be provided, or it should be evaluated if there is maybe an equivalent browser API that hasn't yet been considered. The latter is usually the best approach since it requires the least work.

To test the whole application, you should always use real devices or at least the simulators. For Android, Android Studio with its simulator can be used. For iOS, you should use XCode and the associated simulator (unfortunately only running on Mac OS).

Business Layer:

It is the Controller Layer that handles all the logic. It consists of service classes and uses services provided by data access layers.

Persistence Layer:

It is the Model Layer that contains all the storage logic and translates business objects from and to database rows.

Database Layer:

In the database layer, CRUD (create, retrieve, update, delete) operations are performed.