Alshayun: A mobile content-delivery application

Jacob Chappell

March 29, 2019

University of Kentucky

Introduction

What is Alshayun?

Alshayun is a mobile application for delivering articles consisting of rich text and interactive **applet**s to **reader**s.

Alshayun is designed for three actors in mind (a user is any of them):

Content Author A **content author** is anyone who has anything that they would like to write.

Reader A **reader** is anyone who would like to read what one or more **content author**s have to say.

Developer A **developer** is anyone capable of developing **applet**s and other functionality of **Alshayun** that **content author**s and **reader**s can use.

Live Demonstration

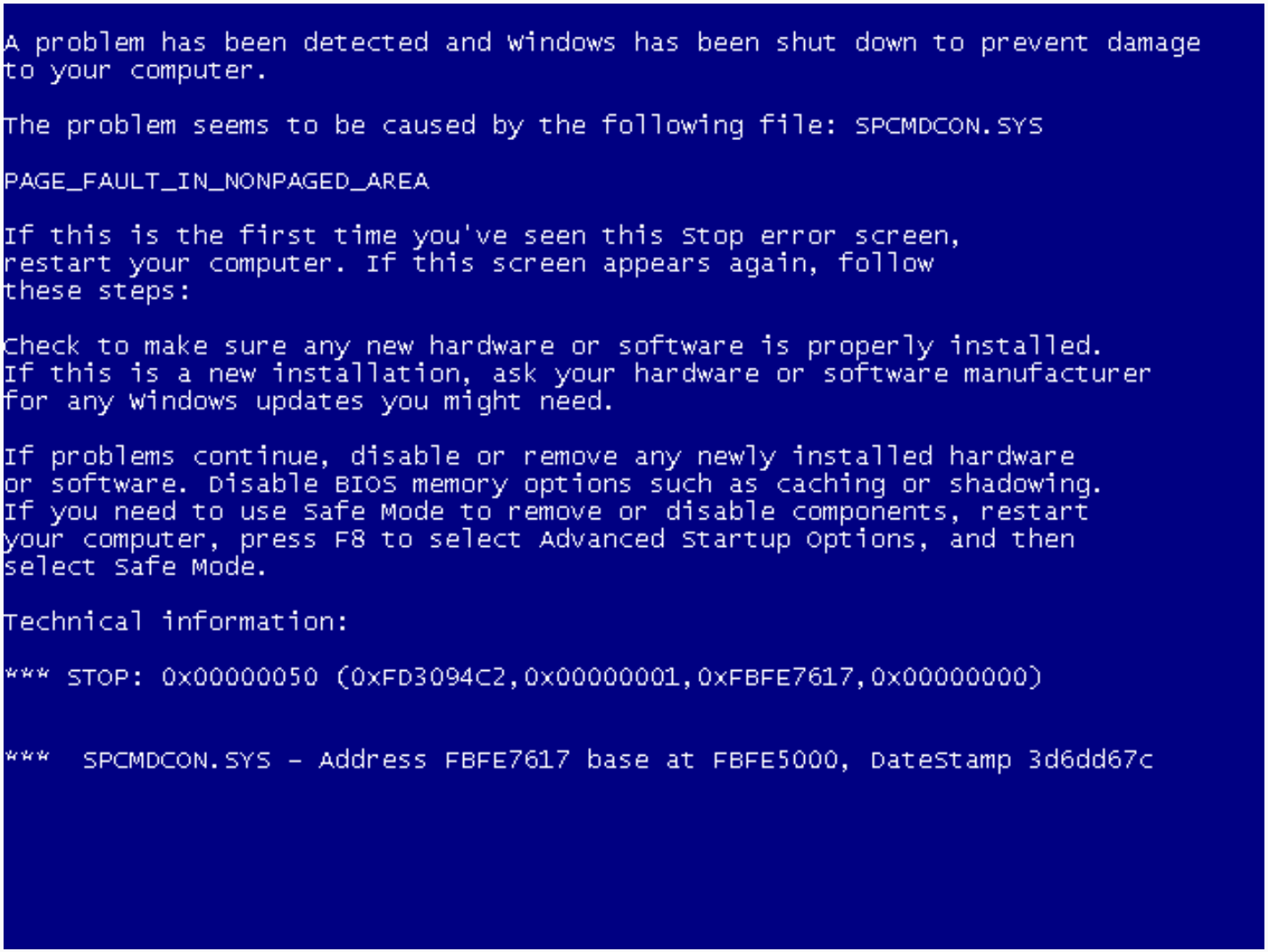

Figure 1: Foreshadowing how this live demonstration will go

Inspiration

My fascination with Bézier curves led me to the Internet.

I discovered the article A Primer on Bézier Curves by Pomax.

I was intrigued by the interactive **applet**s weaved seamlessly throughout the text.

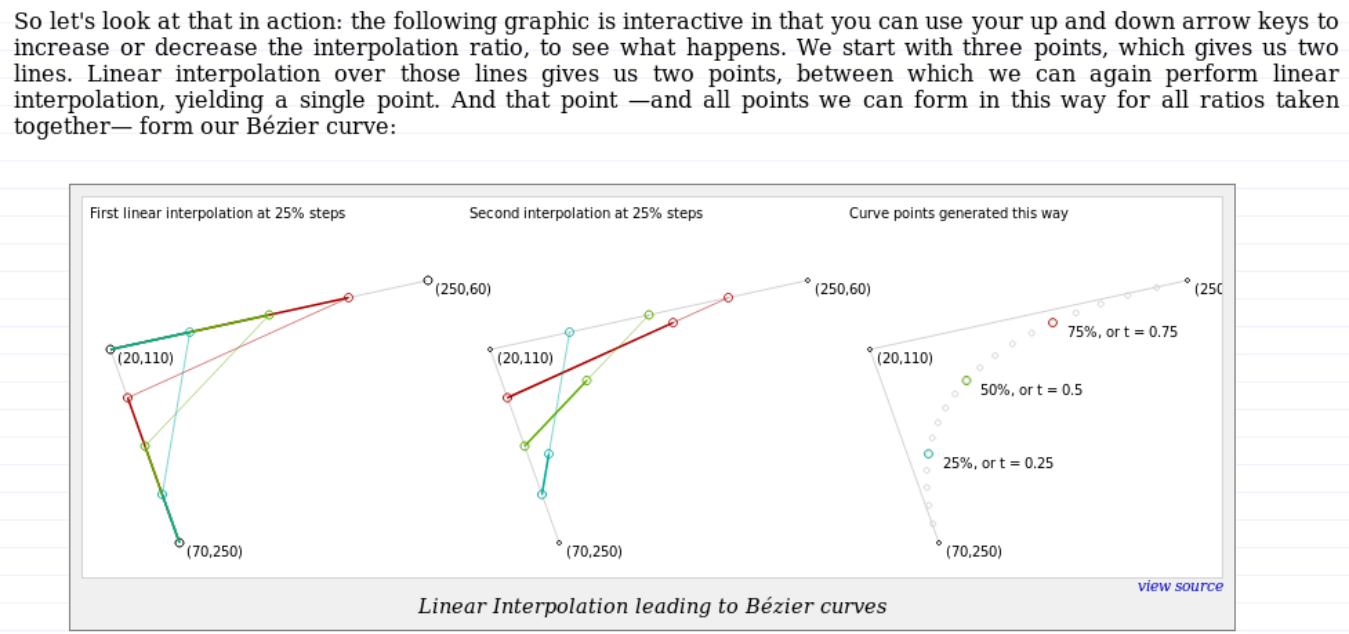

Figure 2: Example **applet**s from *A Primer on Bézier Curves*

About the Name

At inception, **Alshayun** was planned to be a mathematics education platform.

I built a more general platform, but kept the original "mathy" name.

Origins of Algebra

Arabic ("al-jabr")→Spanish→Latin→English

Arabic "al-shayun" appeared frequently in the original algebraic documents and means "the unknown thing"; it became the "x" in Latin/English-speaking algebra (see Why X marks the unknown by Terry Moore).

Alshayun

Architecture

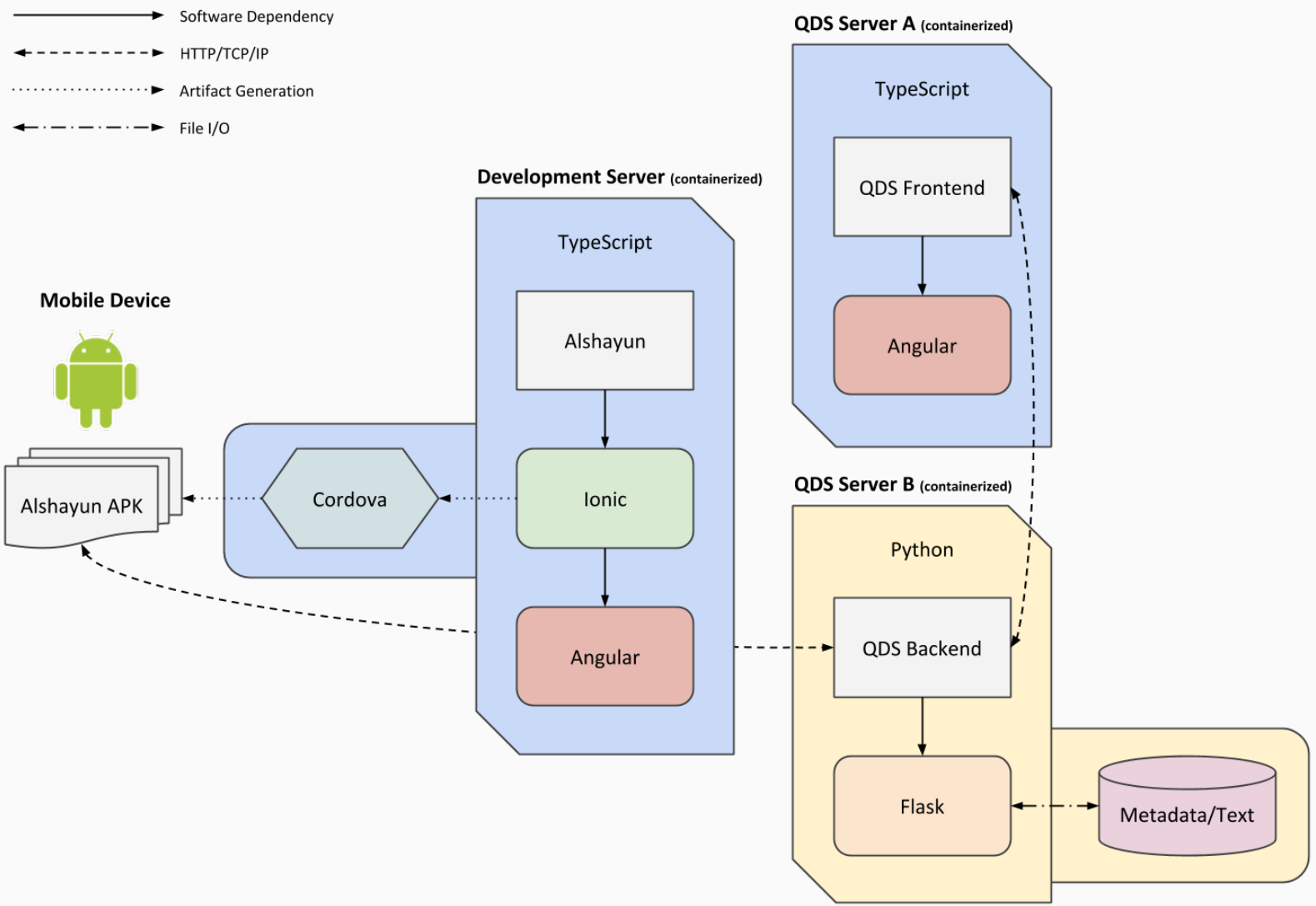

Figure 3: High-level architecture of **Alshayun**

Caching Strategy i

Articles are broken into two pieces: metadata and text.

The metadata of all articles is stored in a single manifest represented in **JavaScript Object Notation (JSON)** and consists of:

id	A unique, monotonically-increasing integer.
title	The title of the article.
tags	Zero or more taxonomic tags.
excerpt	A short (one sentence) description of the article.

Text is the full article text stored separately from the manifest and represented in Markdown.

Caching Strategy ii

```
[
  {
    "id": 12,
    "title": "Towers of Hanoi",
    "excerpt": "Learn about the Towers of Hanoi...",
    "tags": [
      "algorithms"
    ]
  }
]
```

Figure 4: Excerpt of article manifest in **JSON**

Applets i

Applets are the crux of **Alshayun**.

Applets are written in **TypeScript** and implemented as **Angular** components that extend an **Applet** superclass. **Angular** is a **TypeScript** Web framework.

The superclass provides a **Hypertext Markup Language (HTML)** canvas tag and drawing context, implements an animation loop, and provides an extensible toolbar.

Applets are included in articles via an `<applet>` **HTML** tag with extensible properties.

Applets ii

```
<applet name="sort"
        width="50%"
        data-method="insertion"
        data-num-bars="50"></applet>
```

Figure 5: Including an **applet** with an `<applet>` tag

Applets iii

```
@Component({
  template: appletsGenericTemplate + `
    <ng-template #childToolbar>
      <a *ngIf="!doTicks && !allDone"
         (click)="doTicks = true">Play</a>
    </ng-template>
  `,
  styleUrls: ['./applet.scss']
})
export class AppletSortComponent extends Applet {
```

Figure 6: TypeScript excerpt of the Towers of Hanoi **applet** class

Quick n' Dirty Server (QDS)

What is the QDS?

The **QDS** is a stand-alone Web application that allows **content author**s to create and manage articles.

The **QDS** consists of two components: a **backend** and a **frontend**.

The **QDS** is completely containerized with **Singularity** to simplify building and deployment.

Backend i

Written in **Flask**, the **backend** serves two functions: deliver articles to **Alshayun** and expose a **Representational State Transfer (REST)** interface to the **frontend**.

Flask is a **Python** framework for designing **REST**-ful interfaces.

```
@app.route('/articles/<path:filename>')
def get_article(filename):
    return send_from_directory('articles/', filename)
```

Figure 7: QDS interface to serve static articles

Backend ii

```
@app.route('/article', methods = ['POST'])
def create_article():
    article = {}
    article['id'] = checkout_serial()
    # ... finish populating article object ...
    f = open('articles/article.' + \
        str(article['id']) + '.md', 'w')
    f.write(str(request.json['text']))
    f.close()
    # ... add article object to manifest ...
```

Figure 8: Excerpt of **QDS** interface to dynamically create articles

Frontend

The **frontend** is written as a stand-alone **Angular** application.

The **frontend** communicates with the **backend** over TCP/IP via the **REST**ful interface the **backend** exposes. Therefore, the **frontend** and **backend** can be on different servers.

The **frontend** allows for the creation and management of articles.

The **frontend** features a live preview of rendered articles as the **content author** writes them in Markdown.

Production Considerations

The **QDS** is for development purposes and is not production-ready without at least the following additional steps:

- Configure a real Web server (Nginx, Apache) to back the **frontend** and **backend**.
- Use a Web Server Gateway Interface (WSGI) module for interfacing with the **Flask** code.
- Set up Secure Sockets Layer (SSL) certificates and force HTTPS for all communication.
- Develop an authentication wall around the **REST**ful interface of the **backend**.

Motivation

During initial development, I packaged articles into the Android Package (APK) file of **Alshayun**.

This temporary solution hindered the production-readiness of the application.

To improve the quality of the implementation, I set up an Nginx Web server on my desktop and hosted articles there.

I wanted everything necessary to build, run, and test **Alshayun** to be included in the source code with detailed instructions.

The **QDS** was born.

Technologies Used

Node.js

Node.js (Node) is a **JavaScript** runtime designed mainly to facilitate writing scalable, server-side software in **JavaScript**.

Node was first released in May of 2009.

Node has received much attention and both praise and criticism from developers.

The **Node Package Manager (NPM)** was released in January of 2010 to facilitate publishing and installing **Node** packages.

NPM has transcended its original intent to become a general-purpose **JavaScript** package manager.

Alshayun doesn't use **Node** directly but is heavily dependant on **NPM**.

TypeScript

TypeScript is a superset of **JavaScript** that compiles into **JavaScript**.

TypeScript introduces type safety and true object-oriented programming constructs to **JavaScript** such as classes, access modifiers, and inheritance.

The **TypeScript** compiler is available for installation via **NPM**.

Alshayun is mostly written in **TypeScript**.

Angular

- is a Web-application framework developed by Google and written in **TypeScript**.
- is relatively new (ca. September 2016), but it is based on a rewrite of its older predecessor, AngularJS.
- is useful for developing responsive, single-page Web applications.
- comes with a suite of development tools for writing, building, testing, and deploying applications.
- is designed around the **Model-View-Controller (MVC)** design pattern.
- is available for installation via **NPM**.
- provides a built-in development server for prototyping purposes.

Ionic

- is a framework for building cross-platform applications in **HTML**, **CSS**, and **JavaScript**.
- supports developing an application with a single codebase that deploys to Web, Android, and iOS devices.
- is divorced from an underlying **JavaScript** framework as of version 4, but support for **Angular** is strong.
- provides a collection of **HTML** tags and **Angular** components that generate Web components in similar style to Bootstrap.
- is the high-level framework that **Alshayun** is written in, backed by **Angular**.

Cordova

- is an Apache project that provides a uniform interface for generating device-dependent code for Android and iOS.
- provides, e.g., a **JavaScript** interface for interacting with the built-in camera of mobile devices.
- is a vital part of **Ionic** that allows **Ionic** to generate its platform-dependent code from a single codebase.

Flask

- is a **Python** framework for the rapid development of **REST**ful **Application Programming Interface (API)**s.
- allows developers to prefix **Python** functions with decorators indicating the URL endpoint that triggers the function, the acceptable HTTP methods, and more.
- provides a collection of helpful methods for generating HTTP responses, handling exceptions, and processing HTTP request data.
- provides a built-in development server for prototyping purposes.

Singularity

Singularity is containerization software that allows users to develop, package, and relocate full-fledged compute environments consisting of an operating system and software binaries and libraries.

Alshayun consists of three **Singularity** containers I've developed: one for the **Ionic** development environment, one for the **QDS backend**, and another for the **QDS frontend**.

Check out a previous Introduction to Singularity Containers presentation I gave at Keeping Current Seminar for more information.

Architecture

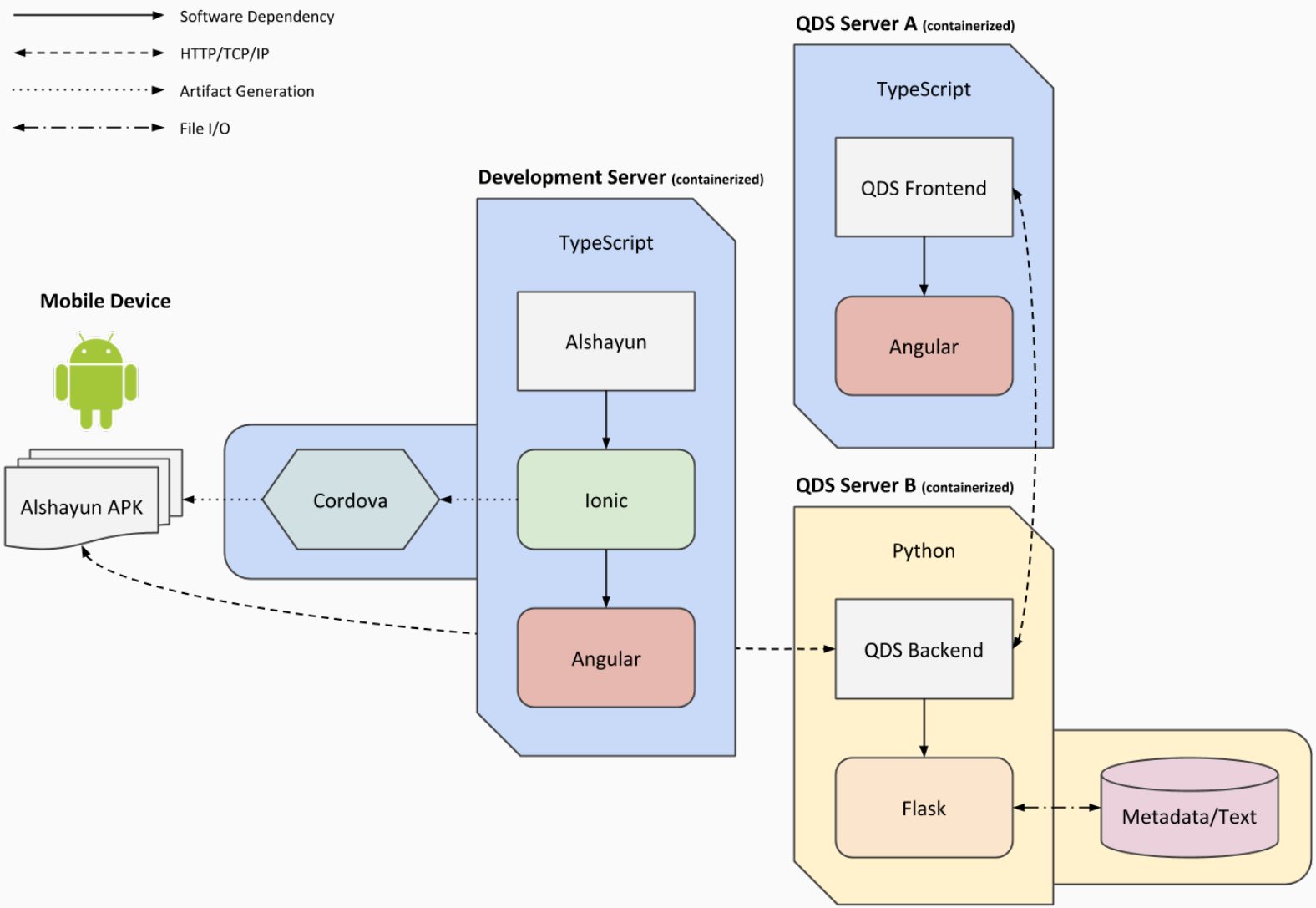

Figure 9: High-level architecture of **Alshayun**

Use Cases

Classroom Auxiliary Content

K–12 schools and universities may find **Alshayun** to be useful.

For example, independent classrooms can run their own articles servers (**QDS**).

Students may be notified of which articles server to use in their copy of **Alshayun**.

Teachers may write helpful articles or lecture notes making use of **applet**s if desired.

Access to articles servers could be restricted to the campus network.

The tagging feature of **Alshayun** could be useful for organizing subjects.

This use case was my main focus when designing **Alshayun**.

Starter Mobile Blog

Alshayun can allow an upcoming blogger to quickly build a blog without building a lot of infrastructure.

Alshayun would be useful for blogs if it were to become a more general and widely used platform like a Really Simple Syndication (RSS) reader.

Future Work

User Accounts

I want to allow **reader**s to save or bookmark articles, comment on articles, and synchronize their personalized settings between devices.

This requires an overhaul of the **QDS** and carries security and privacy concerns.

Accounts need roles so that **content author**s can also write and publish articles and manage access permissions.

Cloud Hosting

A central production **Alshayun** server hosted on the cloud (i.e., Amazon Web Services, Google, Azure) would be nice.

This would be useful as the default articles server.

I imagine this feature could spiral into a larger content delivery platform, although the feature to host a local **QDS** would still be a plus.

Power Efficiency

The **applet**s are power-hungry.

I need to work on more caching and accessing the network only when necessary.

Also, I should unload computationally intensive **applet**s when they are not being used and lazily load as much content as possible without sacrificing too much application performance.

Conclusion

Learning Outcomes

I learned the essentials of **TypeScript**, **Angular**, and **Ionic**.

I fell in love with **Angular**.

My favorite feature of **Angular** was "observables."

The biggest challenge I faced was getting the dynamic **applet**s generated from Markdown working correctly.

I have since begun working on an **Angular**-powered Web site for my mom who is a REALTOR®.

Q&A

Jacob Chappell
jacob.chappell@uky.edu

Source Code and Documentation
https://github.com/phpHavok/alshayun