# りあクト!

Story of React developers

## Firebaseで始める サーバーレスReact開発

大岡由佳



**これが決定版！ この1冊で本格Firebaseアプリが作れる**

Firestore単体での全文検索も紹介！

Seedデータ、Puppeteerでスクレイピング、Reduxを使わない副作用処理などトピック満載

シリーズ3千部突破！

前作BOOTH売上4位

なんとか React エンジニアとして戦力になれた秋谷だったが、既存プロダクトのフロントエンド改修に駆り出される日々に辟易していた。
そんなとき柴崎が持ってきた新プロジェクトの話。Firebase の技術検証のために試験的に二人で自由に開発させてもらえるという。
既存の技術と比べてクセが強く制限も少なくない Firebase に最初は面食らっていた秋谷も、次第にそのパワーに魅了されていく……。

大好評の React シリーズ本、待望の Firebase 編！

## 本書の内容



# リアクト！

Story of React developers

# りあクト!
# Firebase で始めるサーバーレス React 開発

大岡由佳　著

くるみ割り書房

# まえがき

『りあクト！』は TypeScript を始めとするモダンな環境での React 開発を、先輩が後輩に教える設定による対話体で書き下ろした技術同人誌で、これまでに『りあクト！ TypeScript で始めるつらくない React 開発』と『りあクト！ TypeScript で極める現場の React 開発』の 2 タイトルがリリースされました。初出は 2018 年 10 月の技術書典 5 で、2019 年 9 月現在、シリーズ累計の売り上げで 3,000 部を達成しています。国内の React 開発者たちにもかなり好意的に受け入れられているようで、あの会社やあの会社の開発現場で回し読みされてるよ、というような嬉しい噂を筆者もひそかに伝え聞いています。

さて、本書はその『りあクト！』シリーズの第三弾となる Firebase 編です。Firebase、というか BaaS（Backend as a Service）を採用したサーバーレスのフルスタック開発をテーマとして扱おう、というのは実はかなり以前から考えていました。筆者はフリーランスのフロントエンドエンジニアとしていろいろな会社の現場で開発を手伝ってきたのですが、そこで常に悩ましいのがサーバーサイドエンジニアたちとの協業でした。RESTful API をサーバーサイドで用意してもらってフロントエンドからそれを叩くというのが現状よくあるアプリの構成なのですが、どの現場でもサーバーサイドチームとのコミュニケーションの齟齬がどうしても発生しがちです。細かなところでは API 名やメソッドが実態を表していないといったところから始まり、インターフェースやエラーハンドリングがイケてないとか、そのデータ設計は正直どうなのといった不満を抱えつつも、納期を考えると今さら言ってもチームの和を乱すだけなので黙っているようなケースも多々ありました。運良く技術力の高いサーバーサイドエンジニアと働くことができても、都度彼らにお伺いを立て、お願いして準備をしてもらわないとフロントエンドは何もできない羽を縛られたようなもどかしさを、以前は自分もサーバーサイドエンジニアだったためかより一層感じてしまっていました。

さらには RESTful API は今の高機能なフロントエンド開発にとっては柔軟性に欠けるのでは（だからこそ Redux のように高機能な State 管理のしくみが必要になる）といった不満もあり、ベンダーが綺麗なインターフェースを最初から用意してくれていて、フロントエンドエンジニアが自由に開発できる BaaS を使った開発にできればシフトしたいと考えていました。

そこで選択したのが Firebase です。2019 年現在、BaaS の中で最も実績があってサービスの統合性

や使いやすさが抜きん出ている Firebase を採用するのは自然なことでしょう。他に AWS Amplify も検討しましたが、やはり採用事例がまだまだ少ないのと、AWS 自体はメジャーですが Amplify は Firebase 対抗のためのサービスの寄せ集め感が否めず、使い勝手があまりよくなさそうということで見送りました。また、2019 年 2 月に Cloud Firestore がついに正式版リリースを迎え東京リージョンでの提供を開始したことも、Firebase を選ぶ後押しになりました。

でもいざ Firebase を使って開発を初めてみると、やはり React と TypeScript という環境でのサンプルは少なく、あったとしても最新の Hooks を使った書き方などはほぼ見当たりません。また Firebase は公式のドキュメントが分量的にはかなり充実しているのですが、目を通すだけでも大変なのに本格的なアプリを作ろうとするとそこにある情報だけでは難しかったり、また細かなノウハウは動画などに散らばっていたりと、公式ドキュメントさえ読んでおけば大丈夫というものではありませんでした。英語の書籍も何冊か購入してはみたのですが、やはり思ったほどには役に立たず。けっきょく、公式ドキュメントを下敷きに試行錯誤で地雷を片っぱしから踏み潰しながら、そのたびに GitHub Issues や Stack Overflow を当たったり、Firebase の技術サポートに問い合わせたりしながら、何とか業務にも耐えうるレベルのアプリを完成させられるノウハウを蓄積していきました。ですので筆者としてもこれを本にする意義は十分にあると思っていますし、個人的にも備忘録として今後活用していきたい内容です。

なお本書の体裁としては前作までと同じ主人公、柴崎雪菜と秋谷香苗の会社で Firebase を使った試験的プロジェクトが始まり、Firebase 未経験の秋谷が柴崎に教わりながら二人でアプリを開発していく形で話が展開していきます。想定するターゲットの読者としては、React や TypeScript は使ったことがありリレーショナルデータベースについての知識もあるが、Firebase は未経験といった方になります。React や TypeScript は初心者という方は、本署の前にまず『りあクト！ TypeScript で始めるつらくない React 開発 第 2 版』の第 9 章までを読まれることをお勧めします。（「第 9 章まで」というのは、本書では Redux やそのミドルウェアを使わないため）

また本書で扱う Firebase の機能は、Firebase Hosting、Cloud Functions、Cloud Firestore、Firebase Authentication のみとなっています。Cloud Messaging および Cloud Storage その他については触れませんのであらかじめご了承ください。

話の中で柴崎たちが開発しているコミック発売情報アプリ「Mangarel（マンガレル）」ですが、これは本書執筆に当たって筆者が個人開発したものがモデルです。そのままではありませんが、本書の

サンプルコードはそのアプリから抜粋して簡略化したものになっています。アプリについては実際に https://mangarel.com で稼働していますので、概要をつかむため本書を読み進める前に少し使ってみてもいいかもしれません。サンプルコードも https://github.com/oukayuka/ReactFirebaseBook に全て掲載していますので、適宜ご参照ください。

免責のために書いておきますと、技術選定や設計、環境整備のポリシーやプロダクトマネジメントに至るまで本書で提示しているものには筆者の個人的な主張に基づくものが多く、当然ですがこれらが唯一の正解ではありません。あくまで一開発者による意見としてご覧ください。

それでは今回も、二人の会話を通じて Firebase を楽しみながら学んでいただければと思います。

# 本書について

## 登場人物

**柴崎雪菜（しばさき・ゆきな）**

都内のとあるインターネットサービスを運営する会社のフロントエンドエンジニア。React 歴は三年年を超えたところ。本格的なフロントエンド開発チームを作るための中核的人材として、今の会社に転職してきた。当初チームメンバーを集めるのに苦労していたが、初心者の秋谷を迎え入れ React エンジニアとして育てあげた。

**秋谷香苗（あきや・かなえ）**

柴咲と同じ会社のエンジニアでそろそろ新卒三年目。Rails エンジニアとして働いていたが、柴崎の社内メンバー募集に志願してフロントエンドチームに参加。彼女の厳しい指導を経て、今は立派なチームの戦力に。飲み込みの良さと成長の早さは柴崎の折り紙付き。

## 本文中で使用している主なソフトウェアのバージョン

- node 10.16.0 / 12.7.0
- react 16.9.0
- create-react-app 3.1.1
- typescript 3.6.3
- eslint 6.4.0
- @typescript-eslint/parser 2.2.0
- @typescript-eslint/eslint-plugin 2.2.0
- firebase 6.6.1
- firebase-admin 8.5.0
- firebase-functions 3.2.0
- firebase-tools 7.3.2
- react-router 5.0.1
- use-react-router 1.0.7
- commander 3.0.1
- puppeteer 1.20.0
- date-fns 2.2.1

## インストールを推奨する Visual Studio Code の拡張

- DotEnv …… 環境設定ファイル.env のシンタックスハイライトを行う
- ESLint …… ESLint と連携してエディタ内で JavaScript や TypeScript の文法チェックを行う
- Firebase …… Firestore のセキュリティルールおよびインデックスファイルのシンタックスハイライトを行う
- Prettier …… Prettier と連携してエディタ内でコードのフォーマットを行う
- stylelint …… stylelint と連携してエディタ内で CSS の文法チェックを行う
- vscode-styled-components …… styled-components 形式の CSS in JS 文法を stylelint に認識させる

## サンプルコードご利用の注意

本書に掲載されているサンプルコードは、ご自身で各種サービスに登録してそのキーなどを適切に設定しないとダウンロードしてきたままでは動作しません。各章の説明をご覧になった上で、その手順に従って必要な設定を行ってください。

- 1 章以降、プロジェクトルートの .firebaserc.sample を.firebaserc にリネームし、その中のデフォルトプロジェクト ID を読者ご自身が作成した Firebase プロジェクトのプロジェクト ID に書き換える（またはご自身で `firebase init` を実行する）
- 2 章以降、Firebase のコンソールで生成した秘密鍵ファイルを functions/src/ 配下に mangarel-demo-firebase-adminsdk.json として設置する
- 3-3. Rakuten Developers にご自身で登録したアプリの楽天アプリ ID を functions/src/index.ts 内の定数 RAKUTEN_APP_ID に設定する
- 3-4 以降、functions/.runtimeconfig.sample.json を.runtimeconfig.json にリネームし、その中で読者ご自身が登録した楽天アプリ ID を設定する
- 5 章以降、プロジェクトルートの.env.sample を.env としてコピーし、Firebase のコンソールから参照できる API キーやアプリ ID をその中に記述する

なお、サンプルコードは GitHub の以下のリポジトリに章ごとに掲載しています。

https://github.com/oukayuka/ReactFirebaseBook

# 目次

# プロローグ

「はー、そろそろこのプロジェクトの仕事も、元のチームの人たちに引き渡して終わりですね。四ヶ月くらいやってましたっけ。長かったー」

「おつかれさま。でも秋谷さんががんばってくれたおかげで、当初より早めに切り上げられたよ」

「社内中のプロダクトのフロントエンドをモダン化するのに駆り出されるの、いいかげん飽きてきちゃいました。ものづくりがしたくてエンジニアになったのに、ものづくりというより他人の尻拭いばっかりじゃないですか」

「まあ気持ちはわかるよ（笑） でもそんな秋谷さんに朗報。次の仕事は新規のプロダクト開発だから」

「えっ、ほんとですか！？ やったー！ それでどんなプロダクトなんですか？」

「うん。会社の正規のプロダクトというより、技術検証のための試験的プロジェクトなんだけどね。ウチでも Firebase を使っていこうって会社に提言してたんだけど、このたびめでたくそれが通ってこのチームでやらせてもらえることになったの」

「Firebase ですか……。スマホアプリのためのバックエンドサービスでしたよね？」

「いや Firebase はモバイルアプリのためだけのものじゃないよ。ファイルのホスティングもやってくれるし、主要な機能は Web からも使えるようになってる」

「その Firebase を雪菜さんはどうして使いたかったんですか？」

「ふふっ、それはね。秋谷さんも Rails エンジニアだったからわかると思うけど、フロントエンド開発ってサーバーサイドが API を用意してくれないとほとんど何もできないじゃない。でも Firebase があれば、私たちフロントエンドエンジニアだけでプロダクトが最後まで作れてしまう。これってすごいと思わない？」

「あーそのジレンマっぽいの、確かに私も感じてました。Rails で Web アプリを作っていたときは表も裏も自分たちのコントロール下にあって全能感？みたいのがあったんですが、フロントエンド開発は何かやろうとするたびにサーバーサイドにお伺いを立てなきゃいけなくて、あちらの都合に振り回されるんですよね」

「別に誰が悪いわけでもないんだけどストレスたまるよね。それにそれぞれ専門に分かれてチームの

人数も増え、そのぶんコミュニケーションコストが増大するということは、プロジェクトの進行スピードも落ちることになる。いま国内外のスタートアップで Firebase の採用が進んでいるのは、サクッとプロダクトが作れてスピーディーに事業を展開できるからだよね。インフラもほぼメンテナンスフリーで、スケールアップも自動でやってくれるし。だからウチも Firebase を導入することでプロダクトの開発サイクルを短縮して、ヒットを生み出すための手数を増やせるようにしましょう！って言ったらトップの人たちがノッてきたというわけ」

「雪菜さん、偉い人をノセるの上手そうですしね（笑）

　それで、そのプロジェクトではどんなアプリを作ることになってるんですか？　プロダクトマネージャーやデザイナーさんのアサインも決まってるんですよね？」

「検証のための試験的プロジェクトだから、私たち二人以外にメンバーはいないよ？　プロダクトマネージャーは私が兼務。デザイナーもいなくて、Semantic UI をカスタマイズして使う」

「ええ――っ！　雪菜さんがプロダクトマネージャーですか？　できるんですか？　経験は？」

「業務としての経験はないけど、個人開発でならアプリはいくつか作ったことがあるよ。それにまあ、今回はヒットしなくても責任は取らされないので大丈夫でしょう」

「そうか、試験的プロジェクトですもんね。それで肝心のアプリですけど、何を作るつもりなんですか？」

「うん、漫画単行本の発売情報アプリを作ろうと思ってる。私の漫画好きは知ってると思うけど、チェックしてるタイトルが多すぎて、知らない内に新刊が出てるのを見逃してることが多くてね。そういうのを解消してくれるアプリ」

「漫画情報アプリですか……。でももうたくさんあるような気がしますけど。ほら、検索したら Web サービスでもスマホアプリでもこんなにありますよ」

「知ってるよ。でもいろいろ使ってみたけど、どれもピンと来なくてねえ。私は書店のコミック新刊コーナーが好きで、ことあるごとに立ち寄ってて、あそこに行くとすごくワクワクするんだけどそういうワクワク感がどのプロダクトにもないんだよね」

「ん――、ざっとスクリーンショット見ただけですけど、言いたいことはなんとなくわかるような気がします」

「そう！　知ってる？　Google は世界で 12 番目にリリースされた検索エンジンだってこと。Facebook は 10 番目の SNS。iPod のことは誰でも知ってるけど、世界初の MP3 プレーヤーである mpman の名前を知ってる人がどれだけいる？　デファクトスタンダードが確立されていなくてプロダクトが乱

立しているマーケットでは、後発でも切り口次第でいくらでも覇権が狙えるんだよ」
「へー、なるほど。それだけ自信たっぷりに熱弁を振るわれると、雪菜さんならやってくれそうな気がしてきました。ちなみにプロダクトの名前は決まってるんですか？」
「うん、Manga の Release 情報アプリだから『Mangarel』がいいかなって。発音は『マンガレル』」
「シンプルでいい名前ですね。おぼえやすいし、『れる』って語尾が可能動詞っぽいのもなんか幸先よさそう」
「ふふっ、でしょう？　もう mangarel.com のドメインも取っちゃってるしね」
「気が早いですね……。でもすごく楽しそうな仕事ですね。私もワクワクしてきました！」
「お、やる気になってくれたね。来週アタマに Mangarel プロジェクトのキックオフミーティングを入れたので、正式にはそこから開始かな。表向きは試験的プロジェクトだけど、ガチでヒットを狙っていくので、いっしょにがんばりましょう！」
「はい！　よろしくお願いします！！」

# 第 1 章 プロジェクトの作成と環境構築

## 1-1. 基本環境を作る

「じゃあ今日から開発を始めていきましょうか。まずは環境を作らないとね。念のため Node 環境を作るところからおさらいしておこう」

```
$ brew install anyenv
$ echo 'eval "$(anyenv init -)"' >> ~/.bash_profile
$ exec $SHELL -l
$ anyenv install nodenv
$ exec $SHELL -l
$ nodenv install -l
$ nodenv install 12.7.0
$ nodenv install 10.16.0
$ nodenv global 12.7.0
$ npm install -g yarn
$ exec $SHELL -l
```

「.bash_profile のところは、自分が使ってるシェルの設定ファイル名に読み替えてね。秋谷さんは Z shell を使ってるので .zshrc かな」

「あれ？　Node を 12 系と 10 系の二つのバージョンをインストールするんですか？」

「うん、バックエンド処理のために Cloud Function というサーバーレスの関数実行環境を使うんだけど、それがまだ最新 12 系に対応してないの」

「なるほど」

「じゃ、次行くよ。create-react-app で React プロジェクトを作成する」

```
$ npx create-react-app mangarel-demo --typescript
$ cd mangarel-demo/
```

「create-react-app（CRA）がついに正式に TypeScript に対応したからね。--typescript オプションを指定するだけで TypeScript の環境を作ってくれるようになったの」
「おおー、それは便利な世の中になりましたね！」
「TypeScript の設定はそのままでもまあ使えるけど、tsconfig.json にちょっと手を加えておこう」

```
   {
     "compilerOptions": {
       "target": "es5",
       "lib": ["dom", "dom.iterable", "esnext"],
       "allowJs": true,
       "skipLibCheck": true,
       "esModuleInterop": true,
       "allowSyntheticDefaultImports": true,
       "strict": true,
       "forceConsistentCasingInFileNames": true,
       "module": "esnext",
       "moduleResolution": "node",
       "resolveJsonModule": true,
       "isolatedModules": true,
       "noEmit": true,
       "jsx": "react",
+      "baseUrl": "src",
+      "incremental": true
     },
     "include": ["src"],
+    "exclude": ["node_modules", "build", "scripts", "functions"]
   }
```

「baseUrl を src に設定しておくと、import from MyComponent from 'components/MyComponent';のようにインポートを絶対パスで書けるようになるのでおすすめ。これも CRA が最近になって対応したので使えるようになった。あと incremental オプションを有効にすると、再コンパイルがぐっと速くなる。CRA では開発環境サーバはホットリロードが有効になってるけど、その時間が短縮されるから設定しておくといいよ」
「へー、TypeScript も CRA もめまぐるしく進化してますね」
「あとは除外ディレクトリを設定しておく。特にこの後、Cloud Functions のディレクトリをこの配下に置いてコンパイル設定を別にするので、その functions/ディレクトリを除外しておかないと混ざっちゃうから気をつけて」

「わかりました」

「次に、Lint と Prettier を導入しておこう」

```
$ yarn add -D stylelint prettier
$ yarn add -D eslint-plugin-react eslint-plugin-react-hooks eslint-plugin-import eslint-plugin-jest
eslint-plugin-prefer-arrow eslint-plugin-jsx-a11y eslint-plugin-prettier eslint-config-prettier
eslint-config-airbnb
$ yarn add -D @typescript-eslint/parser @typescript-eslint/eslint-plugin
$ yarn add -D stylelint-config-prettier stylelint-config-standard stylelint-order
stylelint-config-styled-components stylelint-processor-styled-components
$ yarn add -D prettier-stylelint
$ npm install -g typesync
$ exec $SHELL -l
$ typesync
$ yarn
```

「うっ、Lint の推奨設定がもりだくさん……」

「現時点でこれくらいは必要かなというものを盛り込んでおいたよ。後からルールを追加すると、既存のコードを修正するのは大変なので、最初から必要なものは入れておいたほうがいいね。ルールを厳格にするのを嫌う人もいるけど、それなりに定評があって開発者コミュニティでよく使われているものばかりなので、毛嫌いせずに受け入れておいたほうがいいと思う」

「最後の typesync ってコマンドは何ですか？」

「ああ、これはね。package.json の中身を調べて、必要な TypeScript の型ファイルがなければ自動で devDependencies に追加してくれるんだよ」

「へ――、そんな便利なコマンドが……。いちいち yarn info @types/hoge で調べて別途インストールするの、面倒だったんですよね。これから使っていこうっと」

「これらのプラグインや推奨設定を有効にするために.eslintrc.js に設定を追加する必要があるんだけど、160 行以上もあって説明するの大変なので、私が用意しておいた完成済みのファイル[*1] を自分のとこにコピっておいて」

「出た、料理番組メソッドですね（笑）」

「stylelint.config.js や .gitignore も同様ね。それから node_modules/ ディレクトリの中身まで ESLint

---

*1 https://github.com/oukayuka/ReactFirebaseBook/tree/master/01-env/mangarel-demo

をかけないように、.eslintignore ってファイルを作って中身にこの 2 行を追加しておく」

```
node_modules/
*.config.js
```

「あと、npm scripts として ESLint を実行できるように package.json に登録しておきましょう。ついでに commit 前にも自動実行されるように。まず以下を実行」

```
$ yarn add -D husky lint-staged
```

「それから package.json に設定を追加ね」

```
     "scripts": {
       "start": "react-scripts start",
       "build": "react-scripts build",
       "deploy:hosting": "npm run build && firebase deploy --only hosting",
       "test": "react-scripts test",
       "eject": "react-scripts eject",
+      "lint": "eslint 'src/**/*.{js,jsx,ts,tsx}' functions/**/*.ts; stylelint 'src/**/*.css'",
+      "precommit": "lint-staged"
     },
     ⋮
     ⋮
+    "lint-staged": {
+      "*.{js,jsx,ts,tsx}": [
+        "eslint --fix",
+        "git add"
+      ],
+      "*.css": [
+        "stylelint --fix",
+        "git add"
+      ]
+    },
+    "husky": {
+      "hooks": {
+        "pre-commit": "lint-staged"
+      }
+    }
   }
```

「はい、おつかれさま。これで Firebase と関係ない部分の環境構築は完了。次は Firebase のプロジェクトを作成して、それを今作った環境に組み込んでいくよ」

## 1-2. Firebase プロジェクトの作成と初期化

「次は Firebase のコンソールからいろいろ設定していくよ。ブラウザで https://firebase.google.com/?hl=ja にアクセスして」
「へー、Firebase の設定ツールって日本語化されてるんですね」
「公式ドキュメントもほぼ日本語化されてるよ。英語が必要なのはサポートへの問い合わせのときくらいかな」
「おー、助かります」
「じゃあ『使ってみる』をクリック。プロジェクト作成画面に飛ぶけど、Google のツール共通で気をつけてほしいのは、秋谷さんも Google のアカウントって盛ってるのひとつだけじゃないよね？」
「はい。四つくらい持ってて、会社でも会社のアカウントとプライベートのアカウントを切り替えて使ってます」
「そうすると、URL 文字列のパスの最初に /u/0/ とか /u/1/ とか表示されてるはず。今も https://console.firebase.google.com/u/0/?hl=ja ってなってるけど。この 0 とか 1 とかって、そのブラウザでログインしたアカウントの順番になってるの。たとえば会社の PC のブラウザで最初にプライベートのアカウントでログインして、二番目に会社のアカウントでログインしてたら、ここを/u/1/にしないといけない。そうしないと、知らず知らずのうちにプライベートのアカウントで会社のプロジェクトを操作してしまうので。まあ、手で書き換えなくてもページ右上の顔アイコンからアカウントは切り替えられるけど」
「確かに、Google Analytics とか会社のツール使ってるときにもうっかりプライベートのアカウントでパーミッションエラーになったりします」
「チームでちゃんと権限管理をしていれば防げるけど、ただこれから色んなツールが URL を提示し

てくるけど、それって基本的に /u/1/ とかついてない URL なのでそのままアクセスすると見られなかったりするから気をつけて」

「はい、おぼえておきます」

「じゃ、正しいアカウントであることを確認したら、『プロジェクトを作成』をクリックね」


「『まずプロジェクトに名前を付けましょう』って言われてますね」

「四文字以上であれば特に制限はないので好きな名前をつけていいんだけど、気をつけたいのはここで**プロジェクト ID** もいっしょに決まってしまうこと。プロジェクト ID は Firebase サービスの中で一意の名前でね。Firebase では Hosting で **.firebase.app** と **.web.app** の二つのドメインが使えるんだけど、プロジェクト ID はその頭につくサブドメインにもなるの。いったん決まったら後から変更できないし、早い者勝ちで埋まっていく」

「えっ、でもプロジェクト ID なんてどこにあるんですか？」

「よく見て。プロジェクト名の下に鉛筆アイコンといっしょに小さく表示されてる」

「え――っ、これですか？　こんなのわかりませんよ！」

「プロジェクト名入力中にプロジェクト ID も連動するんだけど、そのプロジェクト ID がすでに誰かに取られていると、プロジェクト ID だけ末尾に『-f2704』みたいなポストフィックスがつくの。アイコンをクリックすれば編集画面が開いて、誰かとかぶらない限り他の名前に変更できるけど」

「……これは自信を持って見落としますね」

「だよねえ。まあとりあえずここは『mangarel-demo』としておこうか。そのまま『続行』ね」

「次は Google アナリティクスを使うか聞かれてます」

「これは必要になってから後でも設定できるので、『今は設定しない』にしておきましょう」

「ぐるぐる回ってなんか処理してますね。お、『新しいプロジェクトの準備ができました』と表示されました。『続行』をクリックしてと。えっと、今度は『アプリに Firebase を追加して利用を開始しましょう』って言われてます」


「うん。じゃ、丸アイコンの左から三番目『</>』をクリックして Web のアプリの追加に行きましょう」

「はい。…っと、アプリのニックネームを入力しないといけないみたいですね」

「プロジェクト名と区別する必要がなければ、ここも『mangarel-demo』でいいかな」

「その下の『このアプリの **Firebase Hosting** も設定します』っていうのは？」

「Web アプリとして Firebase のホスティング機能を使ってインターネットに公開するかってことだよ。もちろん設定しておきましょう」

「このドロップダウンの中の『新しいサイトを作成』というのは？」

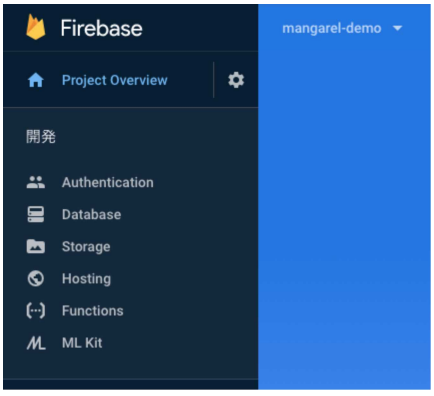
「ひとつのアプリで複数のサイトにデプロイできるんだよ。ここで mangarel-test というサイトを新規に作成すると、このアプリで mangarel-demo.firebaseapp.com の他に mangarel-test.firebaseapp.com にもデプロイできるようになる。最初の mangarel-demo 以外は後で追加したり削除したりできるよ」


「とりあえず今はそのままで行こうか。『アプリを登録』をクリックして」

「ん？『これらのスクリプトをコピーして<body>タグの下部に貼り付けます』とかなんかコードが表示されましたけど」

「ああ、これは Webpack を使う React アプリには関係ない話なので無視でいいよ。あとの ③ ④ のプロセスは単なる説明なので、適当に『次へ』『コンソールに進む』でアプリの追加は完了。

　ついでにデフォルトのリソースロケーションの設定もやっておこう。コンソールのトップに戻ったら、左ペインの上の方に歯車アイコンがあるよね。それをクリック」



「『プロジェクト』欄の中に『Google Cloud Platform（GCP）リソースロケーション』という項目があって、値が『未選択』になってるよね。その鉛筆アイコンをクリックして」
「モーダルウィンドウが開いて、赤字で『設定後にこのロケーションを変更することができません』と警告されてます」


「うん。だからこの設定も慎重にしないといけない。これは Firestore や Cloud Strage を GCP のどのリージョンに置くかという設定なの。Mangarel はもっぱら日本国内向けのサービスなので、日本のリージョンでいいよね。asia-northeast1 は東京、asia-northeast2 は大阪のデータセンターなので、ここでは asia-northeast1 を選んでおきましょう」
「はい、設定完了っと。でもなんだか『プロジェクト』とか『アプリ』とか、一般名称なのに Firebase 用語では特殊な使い方をされていてあまりよくわからなかったです。雪菜さんに言われたままに設定しちゃいましたけど」
「確かにわかりにくいよね。このへんの概念は公式ドキュメント[*2] でまとめて説明されているので、後で目を通しておいたほうがいいかな」
「わかりました！」
「コンソールでの設定はこれで一区切りね。ターミナルに戻って、さっき作ったプロジェクトのディレクトリ mangarel-demo/ にもう一回入り直して以下を実行」

*2 Firebase プロジェクトについて理解する https://firebase.google.com/docs/projects/learn-more?hl=ja

```
$ npm install -g firebase-tools
$ exec $SHELL -l
$ firebase login
? Allow Firebase to collect CLI usage and error reporting information? (Y/n)
```

「この質問は Yes で大丈夫ですよね？」

「サービス向上のための情報収集なので協力してあげましょう。Enter で OK」

「お、ブラウザが立ち上がりました。どのアカウントを使うか聞いてますね」

「Google にひとつのアカウントでしかログインしてなければスキップされる画面だけどね。ここは会社のアカウントを選択しましょう」

「次は『Firebase CLI が Google アカウントへのアクセスをリクエストしています』だそうです」

「それも許可しないと Firebase へのデプロイとかができなくなるのでもちろん『許可』」

「ログイン成功したみたいです。ターミナルも『Success! Logged in as...』って表示されてます」

「よし。ちなみにこのログインセッションは、明示的にログアウトしない限り続くので忘れないようにね。もし別の Google アカウントで別のプロジェクトを持ってて、そちらでデプロイとかの作業をしたいなら `firebase logout` を実行してログアウト、その別アカウントで再ログインしないといけないので」

「わかりました」

「よし、じゃ続けるよ。以下のコマンドで初期化を実行しよう」

```
$ firebase init
```

「なんかかっこいいアスキーアートが表示されて、対話型インターフェースが始まりましたよ？」

「うん、これに従ってさっき作成したプロジェクトを元に React アプリを Firebase アプリとして初期化いくわけだね。今回は Realtime Database や Cloud Storage は使わないので、Firestore と Functions と Hosting のところに矢印キーで移動してスペースキーでチェック、Enter で次に進んで」
「うっ、まだまだ続きますね」
「まとめて示唆するよ。とりあえず次はこれらを選んで。Firestore の設定ファイル名はサジェストされたものをそのまま Enter で OK」

```
=== Project Setup
? Please select an option: Use an existing project
? Select a default Firebase project for this directory: mangarel-demo (mangarel-demo)

=== Firestore Setup
? What file should be used for Firestore Rules? firestore.rules
? What file should be used for Firestore indexes? firestore.indexes.json
```

「次の Cloud Function の設定はちょっと注意が必要。言語に TypeScript を選ぶと TSLint の環境も作成するか聞いてくるんだけど、TSLint はすでに近い将来非推奨になることが決まっているので入れたくない。さらに npm のパッケージをインストールするかも聞かれるけど、TSLint の設定も含めて後でいろいろカスタマイズするので、ここでは断っておいてね」

```
=== Functions Setup
? What language would you like to use to write Cloud Functions? TypeScript
? Do you want to use TSLint to catch probable bugs and enforce style? No
? Do you want to install dependencies with npm now? No
```

「その次の Hosting 設定も変更が必要になる。そのままだと公開ディレクトリが public/ になってしまうので、そこは build を指定する。また SPA として設定するか聞いてくるので、それもデフォルトの No ではなく Yes と答えておくこと」

```
=== Hosting Setup
? What do you want to use as your public directory? build
```

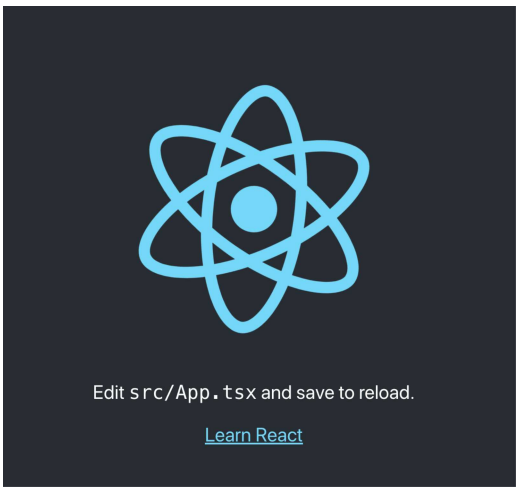
```
? Configure as a single-page app (rewrite all urls to /index.html)? Yes
```

「ふー、やっと終わって『Firebase initialization complete!』と表示されました」
「うん。これは何をやっていたかというと、firebase.json と.firebaserc という二つの設定ファイルを作成してたんだよ。加えて Firestore のインデックスファイル firestore.indexes.json とセキュリティルールファイル firestore.rules も。これらはデプロイのための設定ファイルで、開いてみればわかるけどそんなに複雑な内容じゃない。後から手動でいくらでも変更できる内容なので、いきなりいろいろ聞かれて面食らってたけど、怖がらなくてもいいよ」
「そうなんですね。ちょっとおろおろしちゃいました」
「まだ Cloud　Functions の設定が残ってるけど Hosting の設定は完了してるので、ここでいったんアプリをデプロイしてインターネットに公開してみよう。以下を実行してみて」

```
$ yarn build
$ firebase deploy --only hosting

=== Deploying to 'mangarel-demo'...
✔  Deploy complete!
Project Console: https://console.firebase.google.com/project/mangarel-demo/overview
Hosting URL: https://mangarel-demo.firebaseapp.com
```

「あら、意外とあっさり完了しましたね」
「その表示されてる Hosting URL をブラウザのアドレスバーにコピペしてみて」



「お、いつもの Create React App の画面が表示されましたね。やった！」
「デプロイ成功だね。Firebase は Hosting のデプロイだけならめっちゃ速いからね。フロントエンド部分は、修正したらすぐデプロイというのを気軽に繰り返せるのが嬉しい。ちなみに、.firebaseap.com ドメインだけでなく、https://mangarel-demo.web.app も同時に使えるようになってるから」
「ほんとだ。まったく同じ画面が表示されました」
「よし、じゃ次は積み残しの Cloud Functions の設定を終わらせようか」

## 1-3. Cloud Functions の環境整備

「はい、まずは以下を実行してくれる？」

```
$ cd functions/
$ nodenv local 10.16.0
```

「それから、package.json の以下の部分をこう書き換えてね」

```
    "engines": {
-     "node": "8"
+     "node": "10"
    },
```

「あのー、これは何をしているんでしょうか？」
「Cloud　Functions って、2019 年 9 月現在で正式サポートしてる Node の最新バージョンが 8 系なのね。そのままだと 8 系の環境で稼働してしまう。ただ 10 系もベータ版としてサポートしているので、設定を変更すれば 10 系でも動くの。後で説明するけど、10 系以降でしか提供されいない機能を使い

たいので、10 系を使うように変更してるわけ。自分でも試してみたけど、ベータ版と言っても特に問題は起きてるように見えないので大丈夫だと思う」

「なるほど、わかりました」

「じゃあ、いったん yarn を実行してパッケージをインストールしてみて」

「はい、正常に全部インストールされました！」

「次に TypeScript の設定。tsconfig.json を以下のように書き換えておいてね」

```
  {
    "compilerOptions": {
+     "esModuleInterop": true,
+     "lib": ["dom", "esnext"],
      "module": "commonjs",
      "noImplicitReturns": true,
      "noUnusedLocals": true,
      "outDir": "lib",
+     "resolveJsonModule": true,
      "sourceMap": true,
      "strict": true,
      "target": "es2017",
+     "types": ["jest", "node"]
    },
    "compileOnSave": true,
    "include": ["src"],
+   "exclude": ["node_modules"]
  }
```

「けっこう設定を追加してますね」

「そのままだと import/export 文すら使えなかったり、JSON が読み込めなかったり、Jest が使えなかったり、外部サイトをクローリングしたときに DOM の解釈ができなかったりするからね。

じゃあ次。CRA で作ったプロジェクトのように最初からテスト実行環境が作られていないので、自前で作る必要がある。Jest をインストールして環境を作っていこう。以下を実行してくれる？」

```
$ yarn add -D jest ts-jest @types/jest
```

「どうせ package.json をいじるので、Lint と Prettier もいっしょに設定しようか。これも実行ね」

```
$ yarn add -D eslint prettier
$ yarn add -D eslint-plugin-import eslint-plugin-jest eslint-plugin-prefer-arrow
eslint-plugin-prettier eslint-config-prettier eslint-config-airbnb-base
$ yarn add -D @typescript-eslint/parser @typescript-eslint/eslint-plugin
$ typesync
$ yarn
```

「さらにこれらを使った ESLint の設定を有効にするためにまた長い設定の .eslintrc.js ファイルが必要になるんだけど、これも長くなるので私の設定済みのファイル[*3] をコピっといて。中身はほぼ、ひとつ上の階層の React アプリでの設定からフロントエンドに関する部分を除外したものになってるから。あと .eslintignore ファイルも忘れずに置いておいてね

「わかりました」

「あとは、設定したテストや Lint を実行できるように package.json に設定を追加しておくよ」

```
    "scripts": {
+     "lint": "eslint 'src/**/*.{js,ts}'",
-     "build": "tsc",
+     "build": "npm run lint && tsc",
      "serve": "npm run build && firebase serve --only functions",
      "shell": "npm run build && firebase functions:shell",
      "start": "npm run shell",
+     "test": "jest",
      "deploy": "firebase deploy --only functions",
      "logs": "firebase functions:log"
    ⋮
    ⋮
    "private": true,
+   "jest": {
+     "preset": "ts-jest",
+     "testRegex": "(/__tests__/.*|\\.(test|spec))\\.(ts?|js?)$",
+     "moduleFileExtensions": [
+       "ts",
+       "json",
+       "js"
+     ]
+   }
```

*3 https://github.com/oukayuka/ReactFirebaseBook/tree/master/01-env/mangarel-demo/functions

```
    }
```

「はい、ここまでで Cloud Functions の設定は完了。

なおだけど、ここまで見たように functions/ 配下は同じリポジトリにあるけれども、上の階層の React アプリとは TypeScript や Lint の設定が別々になってるの。だから VSCode でファイルを開く際、フロントエンドを編集するときは開くポイントを mangarel-demo/ に、Cloud Functions を編集するときは mangarel-demo/functions/ にして分けましょう。ビルドプロセスも別々に走らせる必要があるしね。フロントエンドを編集するために開いた VSCode のウィンドウから functions/ ディレクトリを開いて中を触らないほうがいいかも」

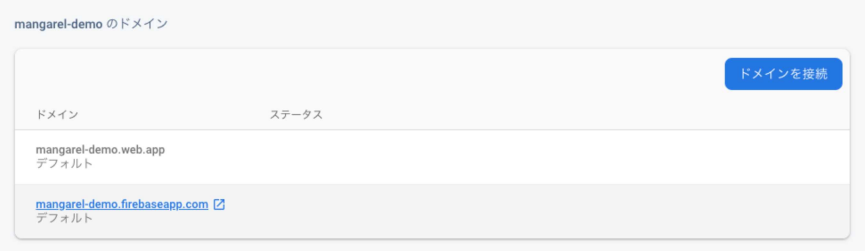
「わかりました」

## 1-4. 独自ドメインを設定する

「とりあえず https://mangarel-demo.firebaseapp.com と https://mangarel-demo.web.app の二つの URL が使えるようになってるわけだけど、これはちゃんとした会社のプロダクトなのでもっとフォーマルな URL を設定したいよね」

「はい。mangarel.com ドメインを取ってたんですよね」

「そう。これからそのドメインを設定していくよ。Firebase コンソールにアクセスして、『Hosting』を開いてみて」



「この青い『ドメインを接続』ボタンをクリック」

「ダイアログが開きました」

「『ドメイン』のところに自分のドメイン、今回なら mangarel.com を入力ね。下の『mangarel.com を既存のウェブサイトにリダイレクトする』はチェックせず『次へ』」


「えーっと、これは何をすればいいんですかね？」

「そのドメインの所有権を持ってることを証明する必要があるの。この『google-site-verification』っていう値を DNS の TXT レコードに設定してあげないといけない。ドメインを登録してるサービスの設定ツールにこんなふうにレコードを設定しましょう」

```
txt @ google-site-verification=**********
```

「伝播に時間がかかるので、5 分ほど待ってから『確認』ボタンをクリック。所有権が確認できたら、今度は A レコードに設定するべき値を表示してくれる。さっきと同様、TXT レコードの記述を上書きする形で設定しておこう」

```
a @ 151.101.??.???
a @ 151.101.??.???
a www 151.101.??.???
a www 151.101.??.???
```

「これで独自ドメイン設定のための作業は完了。ドメインの伝播は５分くらいだけど、SSL 証明書のプロビジョニングには最大 24 時間かかるらしいの。それまではブラウザでそのドメインにアクセスすると『保護されてない通信』扱いになるので気長に待ちましょう」

「えーっと、これでやっと全部の設定が終わりですか？　長かった――」

「うん、おつかれさま。でも私が初めて触ったときはこんなもんじゃなかったよ。コンソールが中途半端に落ちてて、プロジェクトの作成が途中で何度も止まって、けっきょく二日待ちだったし」

「ええっ、そんなことがあるんですか？」

「それだけじゃないよ。それで作った API キーもおかしくて、認証のときに意味不明なエラーが頻出して途方に暮れたもの」

「それでどうしたんですか？」

「Firebase のプロジェクトって、裏では Google Cloud Platform のプロジェクトとして存在してるのね。GCP のコンソールから API キーは再生成できるのよ。コンソール障害時に作成したプロジェクトは Firebase からは見えなかったけと、実は GCP では存在していて無料枠を侵食してたのでそっちから削除したりね。まあそのおかげで Firebase と GCP の関連性を学ぶことができたけど」

「ううん、私ならくじけてそう……」

「Firebase の各サービスの稼働状況は https://status.firebase.google.com から見られるので、おかしいと思ったらそこを確認しましょう。あと、サポート窓口があって回答も迅速なので、どうしてもわからないときはお問い合わせすればいいよ。もちろんやりとりは英語だけどね」

「なるほど、ありがとうございます！」

# 第 2 章 Seed データ投入スクリプトを作る

## 2-1. データベースの作成と Admin 環境の整備

「環境はひと通り構築できましたけど、何から開発していくんですか？」

「そうね。でもその前に、秋谷さんは Firebase まったく初めてだよね。だから当面は、アプリに必要な最低限のラインの各機能を最初に私がプロトタイプレベルで作っていって、それを解説していく形をとろうと思ってる」

「わかりました。早めにキャッチアップしますね！」

「うん、お願い。それじゃあ始めようか。作るのはコミック発売情報アプリだからね。漫画の単行本情報については本の情報はオンライン書店の API から取得するつもり。それで取得した情報をデータベースに格納するのに、各要素をデータモデルに落とし込む必要必要があるんだけと、現時点では必要なモデルは以下の三つくらいかなと思ってるのね」

- **books**（本）
- **authors**（著者）
- **publishers**（出版社）

「ふむふむ」

「それでこのうち出版社については最初からフォーマルなデータを用意しておこうと思ってて、スプレッドシートにコツコツこんなデータを入れてたの」

| | A | B | C | D |
|---|---|---|---|---|
| 1 | id | name | nameReading | website |
| 2 | kodansha | 講談社 | コウダンシャ | https://www.kodansha.co.jp/ |
| 3 | shueisha | 集英社 | シュウエイシャ | https://www.shueisha.co.jp/ |
| 4 | shueishaCreative | 集英社クリエイティブ | シュウエイシャクリエイティブ | http://www.shueisha-cr.co.jp/ |
| 5 | shogakukan | 小学館 | ショウガクカン | https://www.shogakukan.co.jp/ |
| 6 | shogakukanCreative | 小学館クリエイティブ | ショウガクカンクリエイティブ | http://www.shogakukan-cr.co.jp/ |
| 7 | kadokawa | KADOKAWA | カドカワ | https://www.kadokawa.co.jp/ |
| 8 | squareEnix | スクウェア・エニックス | スクウェアエニックス | https://magazine.jp.square-enix.com/ |
| 9 | akitashoten | 秋田書店 | アキタショテン | https://www.akitashoten.co.jp/ |
| 10 | hakusensha | 白泉社 | ハクセンシャ | http://www.hakusensha.co.jp/ |
| 11 | houbunsha | 芳文社 | ホウブンシャ | http://houbunsha.co.jp/ |

「おぉ……、これ作るの大変じゃなかったですか？」

「全部で 76 件しかないから、それほどでもなかったよ」

「なるほど、これを Seed データとして最初にデータベースに入れておくんですね。Firebase にも Rails の `rake db:seed` みたいなコマンドがあるんですか？」

「それがねえ、残念ながらないのよ。だからまずその仕組みを自作するところから始めようと思って」

「自作 Seed スクリプトですか……。普通に必要なんだから、最初から用意してくれてればいいのに」

「既存のデータのエクスポート／インポート機能はあるんだけど、Cloud Storage にバケットとして出し入れするだけで、Rails の Seed のように最初にマスターデータを流し込むといった用途には使えないんだよね。まあないものは仕方ないので、粛々と作っていきましょう」

「はい」

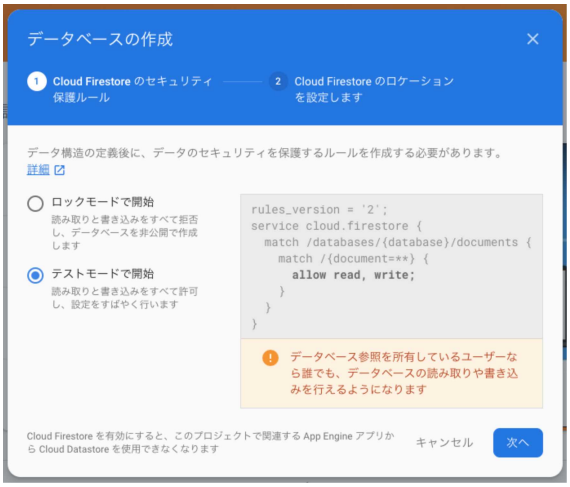
「まずは Firestore を有効にするところからね。Firebase コンソールにアクセスして左ペインの『Database』をクリックして」


「この『データベースの作成』を押せばいいんですよね？　下のほうに『または Realtime Database を選択』ってのが表示されてますけど？」

「ああそれね。Firebase のデータベースには Cloud Firestore と Realtime Database の二つがあってね。Realtime Database のほうが古くて機能的にも見劣りしてるので、そのまま Firestore を選択でだいじょうぶ。Firestore もずっとβ版の時代が長かったけど、2019 年 2 月に正式版になって、こうやってメニューでも標準のデータベース扱いになってるし」

「わかりました。作成をクリックと……。お、モーダルが開いて『① Cloud Firestore のセキュリティ

保護ルール』ってのを聞かれてますけど、どっちを選べばいいですか？」



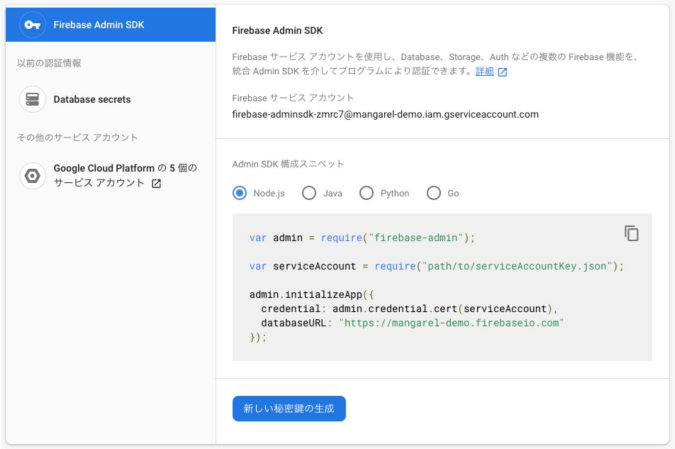

「セキュリティルールについては後々こまかく設定する予定だけど、初期の開発中は全部許可にしといたほうが楽だから『テストモードで開始』を選んで」

「次はロケーションの設定ですね。あれ？『asia-northeast1』以外選べなくなってますけど？」

「これはプロジェクトを作成したときにいっしょに設定したからね。そのまま『完了』でいいよ」

「ぐるぐる回り出した。……っと、作成が完了したみたいですね」

「うん。あと Seed スクリプトは Firebase の Admin　SDK を使うので、その認証のために秘密鍵が必要になるの。左の歯車アイコンから、『プロジェクトの設定＞サービスアカウント』の画面に移動」



「このまま『新しい秘密鍵の生成』をクリックでいいですか？」
「うん、いいよ」
「あ、警告だ。『⚠秘密鍵でプロジェクトの Firebase サービスにアクセスできます。秘密鍵の情報は機密扱いとし、公開レポジトリには保存しないでください』『新しい鍵を紛失すると復元できなくなるため、この鍵は大切に保管してください』。ふむふむ。じゃ、このまま『キーを生成』に行っちゃいますね。お、なんか JSON ファイルがダウンロードされました」
「この mangarel-demo-firebase-adminsdk-*****-*********.json ってのが秘密鍵が格納されたファイルだね。これをプロジェクトルートの functions/src/　ディレクトリに移動させておこう。サンプルコード[*1] では取り回しが悪いので、mangarel-demo-firebase-adminsdk.json にリネームしとくね」
「はい」
「ちなみにこのファイルはさっき警告にあったように絶対秘密のキーで、これがあると第三者がアプリに何でもできてしまうので、ないと思うけどうっかり GitHub の公開リポジトリに入れたりしないようにね」
「……いや、わかってますよ。だいじょうぶです！」

## 2-2. データ投入スクリプトの作成

「Node でコマンドラインインターフェースを実現してくれるライブラリはいくつかあるんだけど、Commander.js[*2] が一番メジャーで TypeScript の型も公式が提供してくれるので、これを使おうと思う。サンプルとして Commander でフェイクの rm コマンドを実装するとこんな感じになるよ」

```
import commander from 'commander';
```

*1　https://github.com/oukayuka/ReactFirebaseBook/tree/master/02-seed/mangarel-demo

*2　https://github.com/tj/commander.js

```
commander
  .version('0.1.0')
  .arguments('<dir>')
  .option('-r, --recursive', 'Remove recursively')
  .action((dir, options) => {
    console.log('remove ' + dir + (options.recursive ? ' recursively' : ''))
  })

commander.parse(process.argv)
```

「arguments() でパラメータを、options() で実行オプションを設定して、それらの値を使って action() の中身の関数で任意の処理を実行する、で合ってるでしょうか？」

「うん、ご名答。このファイル名が rm.js だったとして、実行するときはコマンドラインから node rm.js -r hoge というふうに呼んであげるの」

「なるほど」

「じゃ、必要なパッケージをインストールしていこうか。Commander.js の他に、CSV ファイルをパースしてくれる csv-parse[*3] も入れておくよ」

```
$ cd functions/
$ yarn add firebase commander csv-parse
$ yarn add -D @types/node
```

「ディレクトリ構成だけど、とりあえずこんな感じにしてみようと思う」

```
src/
  commands/
    dbseed.ts
  services/
    mangarel/
      models/
        publisher.ts
      constants.ts
```

*3 https://csv.js.org/parse/api/

「Commander で作成したスクリプトは commands/　の中に置くことにしよう。データベースのモデルファイルは services/mangarel/models/ の中に入れていく。publisher.ts の中身はこうね」

```
import { firestore } from 'firebase/app';

export type Publisher = {
  id?: string;
  name: string;
  nameReading: string | null;
  website: string | null;
  createdAt: firestore.Timestamp | null;
  updatedAt: firestore.Timestamp | null;
};
```

「ID を省略可能にしてるのはどうしてですか？」
「リレーショナルデータベースのレコード ID はレコードの中にあるけど、Firestore のドキュメント（レコードに相当）の ID はドキュメントの外にあるんだよ。Firestore から取得したドキュメントデータはそのままだと ID の値を持たないので、必要に応じて後から入れられるようにしてるわけ」
「なるほど」
「では本丸の Seed スクリプト、dbseed.ts を見てみよう」

```
import commander from 'commander';
import admin from 'firebase-admin';
import fs from 'fs';
import parse from 'csv-parse/lib/sync';

import { Publisher } from '../services/mangarel/models/publisher';
import { collectionName } from '../services/mangarel/constants';
import { addCounter } from '../firestore-admin/record-counter';

import serviceAccount from '../mangarel-demo-firebase-adminsdk.json';

admin.initializeApp({
  credential: admin.credential.cert(serviceAccount as admin.ServiceAccount),
});

const db= admin.firestore();
```

```
const uploadSeed = async (collection: string, seedFile: string) => {
  const buffer = fs.readFileSync(seedFile);
  const records = parse(buffer.toString(), {
    columns: true,
    delimiter: '\t',
    skip_empty_lines: true,
  });
  const ref = db.collection(collection);

  switch (collection) {
    case collectionName.publishers: {
      const docs =
        records.map((record: Publisher) => ({
          ...record,
          website: record.website ? record.website : null,
          createdAt: admin.firestore.FieldValue.serverTimestamp(),
          updatedAt: admin.firestore.FieldValue.serverTimestamp(),
        })) || [];

      for await (const doc of docs) {
        const { id } = doc;
        const docWithoutId = { ...doc };
        delete docWithoutId.id;
        await ref.doc(id).set(docWithoutId);
      }
      await addCounter(db, collection, docs.length);

      return;
    }

    default: {
      throw new Error('specify target collection');
    }
  }
};

commander
  .version('0.1.0', '-v, --version')
  .arguments('<collection> <seedFile>')
  .action(uploadSeed);

commander.parse(process.argv);
```

「このスクリプトのコマンドラインからの使い方はわかる？」

「えっと、コンパイル後の JS ファイルを使うんですよね。`node dbseed.js <コレクション名> <Seed ファイル名>` でしょうか？」

「そう、正解。では処理のほうを見ていこうか。まずさっき作成した秘密キーの JSON ファイルを読み込んで、それを使って Firebase Admin SDK の初期化をしてる。普通の Cloud Functions の場合なら、`firebase login` をしていれば初期化に秘密キーはいらないんだけど、これはコマンドラインから実行する Node アプリケーションなのでこの設定が必要になる」

「ふむふむ」

「それから Commander のパラメータで渡されてきた CSV ファイル、ここではタブ区切りの TSV ファイルにしてるけど、それを読み込んでから csv-parse でパースしてオブジェクトに格納してる。オプションの `columns` に `true` を設定してあるので、最初の行のカラム名がキーになったオブジェクトの配列が生成されてるはず。こんな感じのね」

```
[
  {
    id: 'kodansha',
    name: '講談社',
    nameReading: 'コウダンシャ',
    website: 'https://www.kodansha.co.jp/',
  },
  {
    id: 'shueisha',
    name: '集英社',
    nameReading: 'シュウエイシャ',
    website: 'https://www.shueisha.co.jp/',
  },
  ⋮
]
```

「なるほど。csv-parse 便利ですね」

「その次、Firestore のコレクションというのは RDB のテーブルに相当するものと今は考えてもらっていいかな。コレクションがテーブルで、ドキュメントがレコード。今は publishers だけだけど、今後のことも考えて他のコレクションにも対応できるように作っておいた」

「switch-case のところですね」

「うん。そこでまず、さっき TSV ファイルから取得してオブジェクト配列に変換したものを、Firestore のドキュメントとして保存するための Publisher オブジェクト配列に変換してる。ちなみに、`admin.firestore.FieldValue.serverTimestamp()` というのはサーバ側のタイムスタンプを入れるための

メソッドね」

「その次、見たことない構文ですね。for await of……？」

「そう！　これを使いたいがために Node のバージョンを Firebase がまだβ対応しかしてない 10 系に上げたんだよね。ループの中で await を使うための構文ね。Firestore のコレクションは新規作成直後は、書き込みの上限が秒間 500 回という制限があるので、非同期で一気に書き込みに行くと失敗する可能性があるの。だから 1 回ずつ await で同期的に書き込んでる」

「`ref.doc(id).set(docWithoutId)`ってとこが書き込んでる箇所ですよね。解説お願いします」

「このコレクションリファレンス（publishers）に対して、この ID のドキュメントをこのオブジェクトの値で保存しろってことね。さっきも言ったけど、Firestore ドキュメントの ID はドキュメントの外にあるから、すぐ上でオブジェクトから ID 値を抜き出した後、そのキーと値を除去してるわけ」

「うーん、なるほど」

「ちなみにドキュメントの指定は `db.collection(COL_NAME).doc(DOC_ID)` とメソッドチェーンで書くこともできるし、`db.doc('COL_NAME/DOC_ID')` のようにスラッシュ区切りの文字列で書くこともできる。でも前者のほうがメジャーかな。

あと、大量に一度に書き込みたいときには batch というものが用意されて、それを使うこともできるのね。batch を使って書き込みの部分のコードを書き換えるとこんなふうになる」

```
const batch = firestore.batch();

docs.foreEach((doc: Publisher) => {
  const { id } = doc;
  const docWithoutId = { ...doc };
  delete docWithoutId.id;
  batch.set(ref.doc(id!), docWithoutId);
});

batch.commit();
```

「ふむふむ、どうして今回はこれを使わなかったんですか？」

「batch にも制限があってね、1 回の batch で書き込めるのは 500 件のドキュメントまでなの。さらに `FieldValue.serverTimestamp()` も 1 回にカウントされるので、createdAt と updatedAt にその値を設定する新規作成だと 166 件のデータが限界になっちゃうんだよね。今回の出版社データは 76 件だか

ら問題ないはずだけど、他のデータを扱うときのことも考えて逐次書き込みにしておいたの」
「へー、そんな制限があるんですね。166 件は確かに少ないですね。
　処理の最後に addCounter() という関数を呼んでますけど、これは？」
「うん、Firestore って統計処理に弱くてね。RDB のように COUNT() 一発でテーブル内のレコード数を取得するという芸当はできないの。やるとしたら全部のドキュメントを取得してその数を数えることになる」
「うわ、それは普通に嫌ですね。うーむ、やっぱり RDB とはいろいろ違うんですね」
「どうしても SQL が使いたければ、ドキュメントデータを BigQuery にコピーしてやればいいんだけど、試験的プロジェクトにはそれも大げさすぎるよね。でも各コレクション内のドキュメントの総数くらいは知っておきたいから、こうやってドキュメントの作成・削除のたびに別に用意したカウンターのドキュメントを更新するようにしたの。そのハンドリング関数が addCounter()」
「なんか力業ですね」
「NoSQL の場合、ジョインもできないからあえて非正規化して同じデータを分散して持たせたり、そういう力業みたいのが必要になってくるんだよね。じゃ addCounter() の中も見ておこうか」

```
import admin from 'firebase-admin';
import { collectionName } from '../services/mangarel/constants';

export const addCounter = async (
  db: admin.firestore.Firestore,
  collName: string,
  count = 1,
) => {
  const doc = db.collection(collectionName.docCounters).doc(collName);
  await doc.set(
    {
      count: admin.firestore.FieldValue.increment(count),
      updatedAt: admin.firestore.FieldValue.serverTimestamp(),
    },
    { merge: true },
  );
};
```

「ここのキモは admin.firestore.FieldValue.increment() を使ってるところかな。2019 年 3 月に追加されたばかりの機能で、トランザクションなしにドキュメントのフィールド値のインクリメントがで

きるの」

「へー、便利そう」

「ただこれも制限があってね。Firestore は同じドキュメントを書き換えられるのが秒間 1 回までという制限があるんだけど、そのままこのインクリメントにも適用される。だから、たとえばサービスが超ヒットして 1 秒間に新規ユーザーが 2 人以上登録があったりするとユーザー数の統計値がずれたりする可能性があるんだよね。でもそのときになったらトランザクションで書き直すとか、BigQuery を導入するとかでいいかなって。当面はこれでいきましょう」

「そうですね。サービスの根幹とは関係ない部分ですし、最初からそんなに気合い入れて作り込まなくてもいいと思います」

## 2-3. npm scripts として登録する

「TypeScript のコンパイルは yarn buid でできるようになってるわけだけど、React アプリのホットリロードのように随時コンパイルが走るようにしときたいじゃない？」

「うーん、まあそうですね」

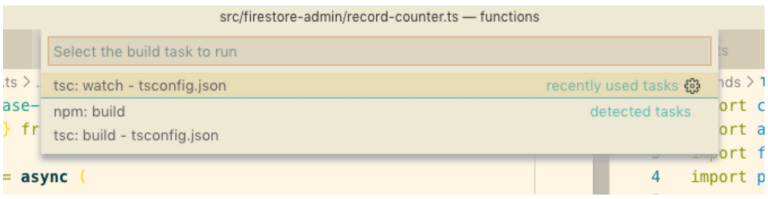
「方法はいくつかあるんだけど、VSCode で tsc のウォッチプロセスを走らせるのが一番手軽でいいと思う。functions/ をポイントにして開いた VSCode のウィンドウで Command + Shift + B を押してみて」

「『Select the build task to run』って選択肢が三つ表示されました」

「その中の『tsc: Watch -tsconfig.json』ってのを選択してみて」

「お、VSCode の下ペインにターミナルが開いて何か表示されましたね」

「tsc のウォッチタスクが稼働したの。『Found 0 errors. Watching for file changes.』ってあるから、エラーなくコンパイルができたみたいね。functions/lib/ ディレクトリの中を見てみて」



```
$ ls lib/
commands/                                  mangarel-demo-firebase-adminsdk.json
firestore-admin/                           services/
index.js                                   utils/
index.js.map
```

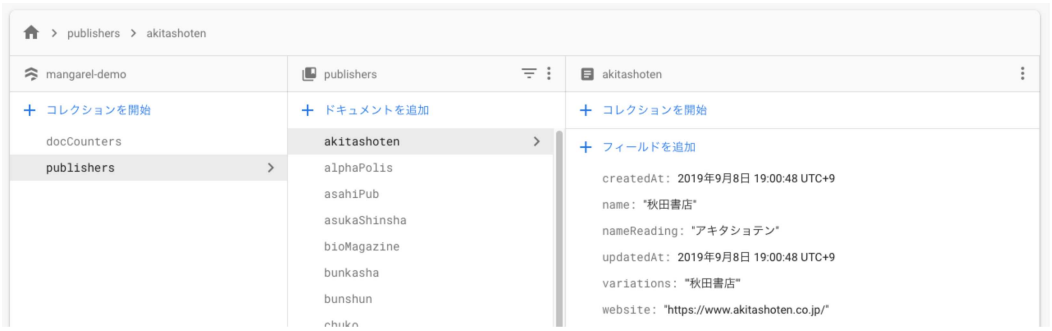
「これはコンパイルされた JavaScript ファイルですか？」

「そう。tsconfig.json のコンパイラオプション `outDir` が"lib"に設定されてるからここに出力されるのね。さっきの dbseed.ts はこの中の commands/dbseed.js として出力されてる。じゃこれを実行してみようか」

```
$ node lib/commands/dbseed.js publishers seeds/publishers.tsv
```

「特にエラーは起きませんでしたが、ちゃんと入力されたんでしょうか？」

「コンソールを見に行ってみよう。Firebase コンソールを開いて、左ペインの『Database』をクリックしてみて」




「おおー、ちゃんと入ってますね。へー、こんなふうにコンソールからデータが確認できるんですね」

「ちょっとした phpMyAdmin みたいなもので、ドキュメントの追加や削除、各フィールド値の変更もできるよ」

「それは便利ですねえ」

「コレクション欄右上の逆三角形のアイコンから、クエリーを発行して絞り込むこともできるよ。ただし気をつけないといけないのはすぐ右隣の縦にドットが三つ並んだアイコンね。ここからそのコレクションの全てのドキュメントがカジュアルに全削除できてしまうの。ぼーっとした頭のまま、個別のドキュメント削除とまちがえて本番稼働中サービスのコレクションの全ドキュメントを削除してしまった人の話もある」
「ひいいいっ、悪夢ですねそれ。胃がキューってなります……」
「本来ならちゃんと管理ツールを自前で用意したほうがいいんだろうけど、最初はなかなかそこまで手が回らないよね。だから特にコンソールからの編集、特に削除は気をつけないといけない」
「わかりました。肝に銘じます」
「じゃ次は、そのドキュメントの一括削除を実際にやってみよう。今言ったようにこのコンソールから GUI で削除することもできるんだけど、誤操作の可能性もあるので一括削除は GUI からは行わないようにしたい。公式からも大量のデータの削除は Firebase CLI から実行することが推奨されてる死ね。じゃ、コマンドラインで以下を実行してみてくれる？」

```
$ firebase firestore:delete publishers
? You are about to delete all documents in the collection at publishers. Are you sure? (y/N) Yes
Deleted 75 docs (Infinity docs/s)
```

「お、コンソールから publishers のドキュメントが全部消えましたね」
「うん。一括削除は SDK にも用意されてないので、こうやって常に CLI からやるようにしましょう。firestore:delete の usage はこんな感じ」

```
firebase firestore:delete [options] <<path>>
Options:
  -r, --recursive
  --shallow
  --all-collections
  -y, --yes
```

「Firestore はコレクション＞ドキュメント＞コレクション＞ドキュメント＞…と親子階層を持てるようになってるんだけど、普通に親ドキュメントを削除しただけでは、その子コレクション（サブコレ

クション）は残ってしまうのね。親子まとめて削除するためには `-r,--recursive` オプションを指定してあげる。あと Firestore 内の全てのコレクションとドキュメントをまとめて削除したいときは `--all-collections` を指定。また、さっきみたいに Yes/No を聞かれずに問答無用に削除したいときは `-y,--yes` オプションを指定してあげれば OK」

「ふむふむ」

「で、毎回これらの長いコマンドを入力するのはだるいので、npm scripts としてもっと短いコマンドで実行できるようにしておこう。package.json を以下のように変更して」

```
    "scripts": {
      "lint": "eslint 'src/**/*.{js,ts}'",
      "build": "npm run lint && tsc",
      "serve": "npm run build && firebase serve --only functions",
      "shell": "npm run build && firebase functions:shell",
      "start": "npm run shell",
      "test": "jest",
+     "dbreset": "firebase firestore:delete --all-collections",
+     "dbreset:publishers": "firebase firestore:delete -y --shallow publishers",
+     "dbseed": "npm run dbseed:publishers",
+     "dbseed:publishers": "node lib/commands/dbseed.js publishers seeds/publishers.tsv",
      "logs": "firebase functions:log"
    },
```

「はい、じゃさっそく実行してみよう。以下のコマンドね」

```
$ yarn dbreset
? You are about to delete THE ENTIRE DATABASE for mangarel-demo. Are you sure? (y/N) Yes
$ yarn dbseed
```

「完了したみたいですね。ちゃんとコンソールでも Seed データが保存されてることを確認できました！」

「うん。今回は publishers だけだったけど、他に Seed データが増えたときはこんなふうに追加していけばいいよ」

```
    "dbseed": "npm run dbseed:authors && npm run dbseed:publishers",
    "dbseed:authors": "node lib/commands/dbseed.js publishers seeds/authors.tsv",
    "dbseed:publishers": "node lib/commands/dbseed.js publishers seeds/publishers.tsv",
```

「なるほど、これは便利ですねー。まさに Rails の `rake db:seed` って感じ。他のプロジェクトでもこの仕組みは使い回せますね！」
「うん、そうしていくつもり。拡張性を考えてかなり汎用的に作ってあるからね」

# あとがき

まえがきでもふれた通り、登場人物の二人が開発しているという設定のコミック発売情報アプリ「Mangarel（https://mangarel.com）」は、本書の執筆に当たって筆者が個人開発して今まさに稼働している実在のアプリです。執筆のために簡単なサンプルコードはこれまで書いてきましたが、一本のアプリを作るというのは初めての試みでした。なぜそうしたかというと、私自身が Firebase 開発の業務経験がなかったので、技術書を出せるくらいの知見を貯めるためにはそれなりの規模のアプリを自分で作ってみる必要があったのと、各機能ごとにそれぞれ関連のないサンプルコードを散発的に書くよりも Firebase のより実践的な本として価値を持たせられると考えたからです。しかし個人開発と執筆を並行して進めるのは想定していたよりずっと大変で、当初はアプリを PWA 化してその内容も本書に盛り込むつもりだったのですが、さすがに手が回らず早い段階で断念してしまいました。というわけで、エピローグにもあったようにこの話は PWA 編として続きます。実質、本書の Firebase 編が前編で、PWA 編が後編となります。順調にいけば、次回の技術書典 8 で PWA 編が出せると思います。

Mangarel は本書の執筆のために開発したと書きましたが、実は構想はずっと前からありました。資料では 2013 年に最初の痕跡があって、mangarel.com のドメインも取り、当時は Scala で開発しようとテーブル定義や出版社一覧などのデータも作ったのですが、だんだんとやる気を失ってフェードアウト。今回の Firebase をテーマにした執筆に当たって、そのアイディアを掘り起こして失効していたドメインも取り直し、なんと 6 年越しで人生初の個人開発アプリとして完成させることができました。プログラマの三大美徳のひとつである「怠惰」を人一倍そなえている筆者にとって、技術書典の締め切りは自分を追い立ててくれる貴重な存在です。

そして今回扱った Firebase。本書の中でも二人に代弁させていましたが、まさにフロントエンドエンジニアをエンパワーする技術だという感想を持ちました。AWS が代表するクラウドサービスが充実することでスタートアップの敷居が下がり、その結果たくさんのユニコーン企業が生まれる状況につながりましたが、Firebase のような BaaS が普及することはそのトレンドにさらに拍車をかけ、個人レベルにまで到達させるのではという期待があります。アメリカでは IndieHackers という個人開発者が成功したストーリーをシェアするサイトが盛り上がっていて、個人開発の世界が活気を帯びてい

ます。まだ億万長者になったというほどの華々しい成功例は少ないようですが、知見がシェアされ、それぞれの力が相互作用してたくさんの化学変化が生まれる環境ができれば、大成功を収めるケースがこの先たくさん出てくるかもしれません。

「個人開発」という言葉は、最近日本でもよく聞くようになりました。そして地味ながらもそこで生活できるだけの収入を得て、専業の個人開発者になっている人も今ではそれほどめずらしくない存在です。そしてそんな個人開発にうってつけなのが Firebase です。今回、アプリの個人開発に Firebase を使ってみてそう実感しました。インフラのことをほとんど気にする必要がなく、もしヒットしても勝手にスケールアップしてくれて、しかも AWS などと比べてもトータルで全然安い。API もほぼ自分で作る必要がなく、セキュリティルールさえしっかりしていればクライアントから柔軟にクエリーを実行できる。個人開発や小規模スタートアップの味方となる、まさに破壊的イノベーションと言えるでしょう。（保守的な、枯れた技術の開発者たちには、せいぜい今のうちに「おもちゃ」だと笑わせておきましょう）

今回、本番稼働している Mangarel からそのままではありませんが、サンプルとしてかなりのコードを気前よく抜粋してきました。これを下敷きにすれば、簡単な情報アプリでならば一週間もかからず作れてしまうのではないでしょうか。これから個人開発をやってみたいと考えている方は、ぜひ活用してみてください。そして私は個人開発にこそ、コードをシンプルに保てる React を使うべきだとも思います。今回、複雑で難解というイメージから避けられがちの Redux を使わなかったのは、React 開発の敷居を下げたいという筆者の気持ちからでもあります。

なお今回の表紙イラストは、雪菜さんと秋谷さんの談笑シーンです。これまでひとりずつピンの構図で来ていたのですが、前作の『りあクト！ TypeScript で極める現場の React 開発』で信頼が生まれた二人がいっしょのところをどうしても載せたくて、イラスト担当の黒木様にお願いしました。その結果、またまたこんなすばらしい絵を仕上げていただき、毎回感謝にたえません。

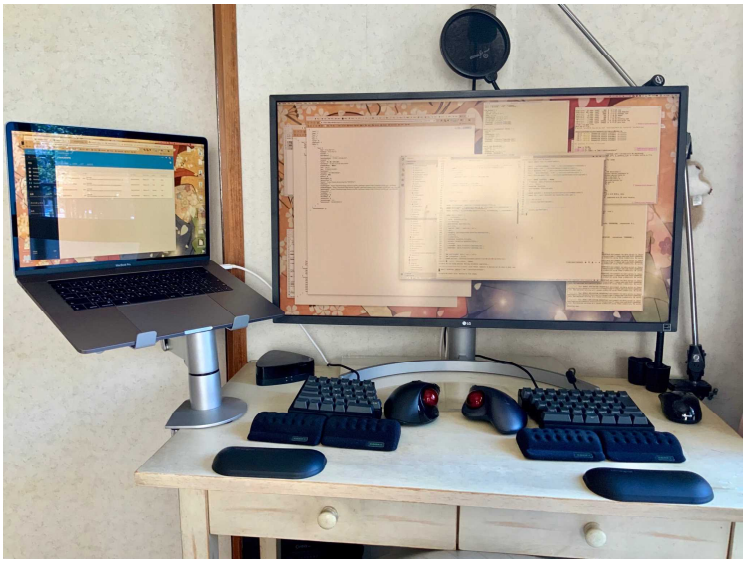
本書の内容についてのご意見・ご感想、またはご指摘などがあれば Twitter やブログなどで言及していただけると、筆者としてはとても励みになります。また、Twitter ハッシュタグ #りあクト をつけてツイートしていただくと、必ず目を通します。なおイラストについてのご感想もお待ちしてます。

それではまたこの次、PWA 編でお会いしましょう。

# 著者紹介

## 大岡由佳（おおおか・ゆか）

インディーハッカー／技術同人作家で分野は React。フリーランスとして様々な現場で開発の手伝いをしていたが、直近ではマーケットから直接稼ぐ収入のほうが多くなったので、フリーランスを名乗るのをやめた。漫画読みで、最近では本書執筆中のあいまに『ダイヤのA』全 47 巻を再度読破。さらに漫画好きが昂じてコミック発売情報アプリ Mangarel を開発する。Twitter アカウントは @oukayuka。



**黒木めぐみ（くろき・めぐみ）** ◎表紙イラスト

漫画家、イラストレーター。

# りあクト! Firebase で始めるサーバーレス React 開発

2019 年 9 月 22 日　初版第 1 刷発行

著者　大岡由佳

印刷・製本　日光企画