サイコロの旅アプリ開発記 #2

~行先不明の旅行アプリを1ヶ月で個人開発して友達を対馬に行かせた話(のつづき)~

kujiro

機能を簡単に説明すると？

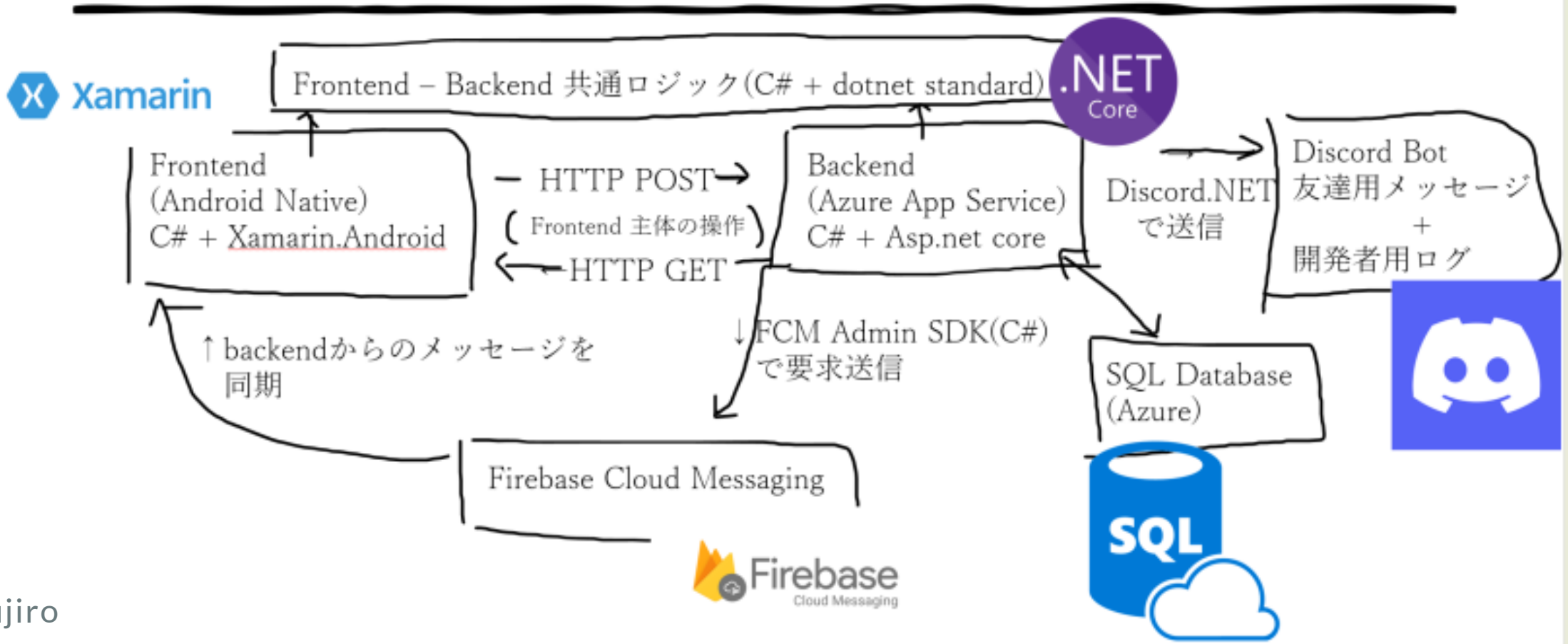
- 行き先不明の旅をしたい友人数名のために専用で開発した旅行支援アプリケーション
- アプリが直前のタイミングで、サイコロ形式で次の行程を逐一発表
  - どこに行くのか最後までわからない旅行を実現
    - 本当にどこに行くのか教えてない
- スマホアプリとバックエンドAPIから構成

技術構成

- フルスタック構成
  - Androidアプリケーションのフロントエンド
  - バックエンドAPI
  - ともに C# で記述
    - コードメトリクス読みで数千行

今回は主に私の技術スタック的な都合により、すべて C# で記述をしました。結構なんでもかけちゃうんですよね。

構成(雑)


Xamarin
Frontend – Backend 共通ロジック(C# + dotnet standard)
.NET Core
Frontend
(Android Native)
C# + Xamarin.Android
HTTP POST
Frontend 主体の操作
HTTP GET
Backend
(Azure App Service)
C# + Asp.net core
Discord.NET
で送信
Discord Bot
友達用メッセージ
+
開発者用ログ
↑backendからのメッセージを
同期
↓FCM Admin SDK(C#)
で要求送信
SQL Database
(Azure)
Firebase Cloud Messaging
Firebase
Cloud Messaging
SQL

開発期間

7月末 ~ 8月末までの **一か月** (!)

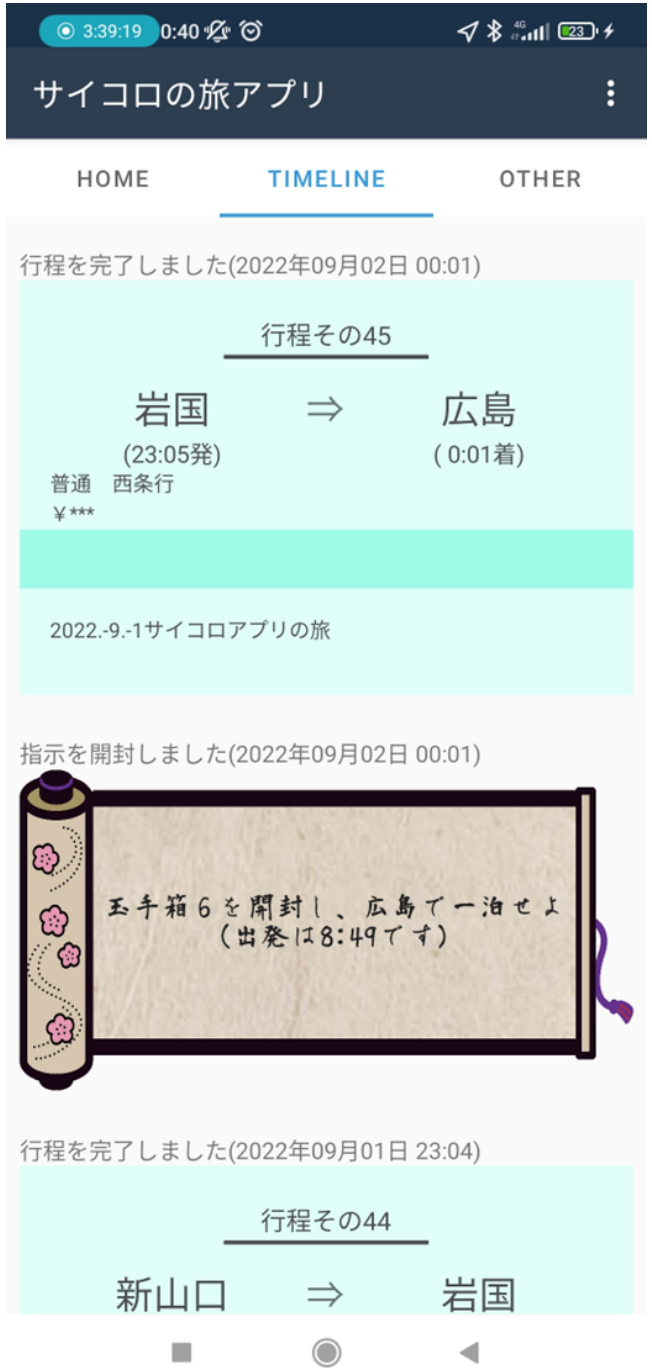
正直、この規模感のアプリケーションを設計から本番運用まで持っていくのに十分とは言い難い期間でした。開発期間の多くは夏休みということで、本当に一日の大半を本アプリケーションの開発に投入できましたが、それでもかなり厳しいスケジュールだったことは事実です。

2022/10/14 HUIT LT会

動作デモ :ホーム画面

現在の行程を確認できるホーム画面の様子。
次の行程を確認できる場合はここから画面遷移できる。
旅行アプリということで、きっぷ風のレイアウトを採用。

kujiro



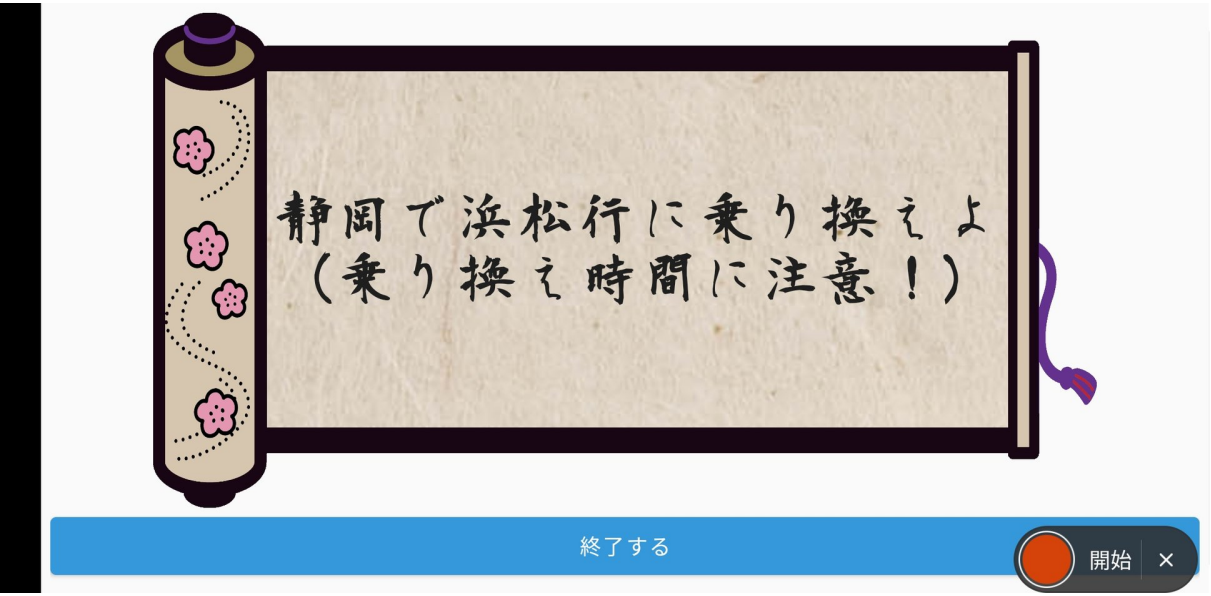
2022/10/14 HUIT LT会

タイムライン

過去の行程を時系列で確認できるタイムライン画面の様子。
見逃し確認や旅行の振り返りで使用することを想定。

kujiro

行程発表



行程発表のテキストを確認する画面の様子。
スクリーンショットでは伝わらないですが、開封時にはアニメーションがつくなど、面白さを意識した構成。

行程抽選(サイコロ)

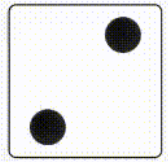
次の行程を抽選する画面の様子。
ここからサイコロを振るCG画面に遷移し、行程が確定する。
次の画像はアニメーションgifとなっています。

2022/10/14 HUIT LT会

名古屋からの進行方向

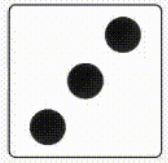
引き続き乗車 北西へ

東海道線　大垣方面



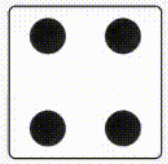
ガンガンいこうぜ

東海道線　大垣方面



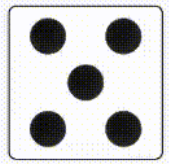
ちょっと休憩　買い物タイム

名古屋で途中下車



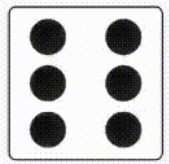
そろそろ降りたい

名古屋で途中下車



伊勢湾をぐるっと南西に三重へ

紀勢本線　津方面



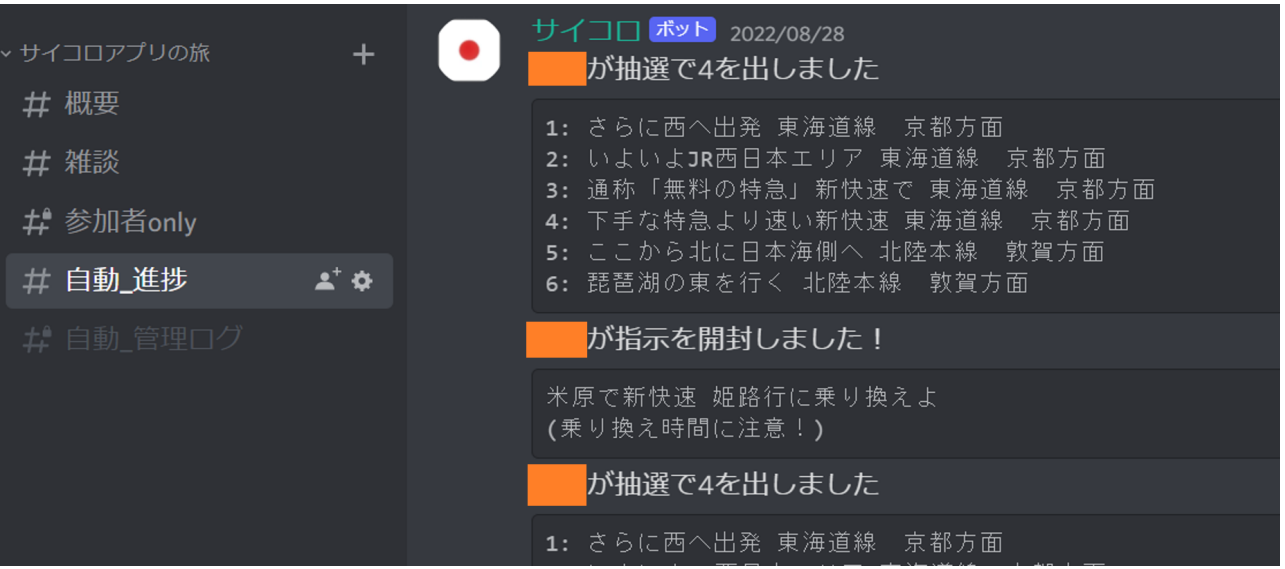
山越えのローカル線で北へ

高山本線　高山方面

抽選を実行する

kujiro

Discord連携



サイコロを振った結果は、discordに自動的に送信される。
これにより、参加者だけでなく、残念ながら都合が合わなかった人や私も旅の行く末を見守ることができる。

つらいとこ#1: 複数人連携

- 最初1人だった
  - その1人が同伴者を集めて4人に
- サイコロを振って目的地を決めるシーンがある
  - 4人の結果が同じになるよう制御しないといけない
- バックエンドが必要になる
  - 進捗の同期も必要に

つらいとこ#2: 通知

- 直前まで「次の行程」を秘密にしたい
  - 10分前に次の行程を発表する
- 通知を飛ばすことでこちらの望むタイミングで開いてもらう
  - フロントエンドがandroidネイティブになった要因
- 飛ばない・10分以上遅れたが起きると致命傷
  - ちゃんと飛ばないとあかん

つらいとこ#3: テスト困難性

- 6日間の旅行の行程を時間に沿って発表するアプリ
  - 実時間依存性が強い
  - しかも6日間!
- ちゃんとUI表示は切り替わる？通知は飛ぶ？
  - でも開発期間は全部で1カ月しかない！
- この辺はアプリの設計で緩和したい
  - DIで時間への依存関係を解消

設計

- 個人開発でもこの規模ならちゃんと設計しましょう
  - 別に受注開発のような「≒コード」みたいな設計をするわけではない
- 最初に設計で決めるべきこと
  - 要件: 何を対応して、何を対応しないか
  - 外部IF: 外部のAPIやライブラリOS機能との連携部分
  - データモデル: 中心的なデータのモデル化

要件の例: オフラインでも動作すべき？

- 旅行アプリなので移動が多い
- しかも行程はアプリでしかわからない
  - 対応はすべき
- 発生確率はどう？
  - そんなに大きくはない

詰むとまずいが、運用で対応してもよさそう(結論)

外部IFの例: android通知API

- 通知が大きなウェイトを占める
  - androidの通知APIに束縛される
  - ので事前に仕様を調べ、こちらの仕様に組み込む
- 何らかの方法で自動起動して、通知APIを叩くようだ
  - `AlermManager` によるタイマベースの起動
    - 端末の省電力との兼ね合いで制約が色々
    - 正確に起動する `setexactandallowwhileidle` は9分に一回
  - FCMからだと制約は緩い

データモデルの例

今回のアプリで出てくる中核となるデータ

- 行程 (何時にどこからどこへ向かうか)
- 指示 (時間になると見れる、次に何をすべきかというテキスト)
- 抽選 (次の行程を決めるためのサイコロ)

の構造と関係性を定義する(≒クラスを書く)

進捗管理

- 限られた時間ゆえ進捗管理が重要
  - 適当なmarkdown作ってタスクを管理
- やりたいこと全部実装する時間はない
  - 適度に優先度付けする

優先度付きキューにやりたいこと突っ込んでいって自分が無限ループで一件ずつ取り出していくイメージ

別のLTで話したこと

こんな感じで設計が完了し、色々苦戦しながら実装完了。
このへんの話もしたいが割愛。

もちろん、実際の旅行を司るアプリなので **一発勝負**
1カ月、夏休みのほぼ全空き時間を投入して開発したが **一発勝負**
ちゃんと運用しないといけない！
(そのためのロギングなども実装した！)

個人開発だけど本気で監視・運用した
　...という話を別のLTでしました(ので割愛)

感想

- 限られた時間ゆえ本気で開発しきった
- しかもめっちゃフルスタック

圧倒的成長!!

- でもあくまで一度限りの使用
  - 少し寂しさはある
    - 前回も言ったが「**お祭り精神**」

ご清聴ありがとうございました

余談:ちなみにこのスライドは markdownで書いてみました

割といい感じ
コードも埋め込める！(パワポだとこれ辛い...)

```
static void Main()
{
    const string message = "thank you for listening!";
    Console.Writeline(message);
    return;
}
```