Mobile Apps
Deloitte Digital Community

Server side Swift

Deloitte.
Digital

Swifting your way into the server-side world

Radu Dan

On the road to Swift 6

Accelerate growth of the Swift software ecosystem

Together we [the Swift community] are working to build a programming language to empower everyone to turn their ideas into apps on any platform.

As a community there are a variety of ways we can accelerate that growth, widening Swift's availability and impact to everyone.

Here are some concrete goals we can work on together as a community:

- **Expand** the number of platforms where Swift is available and supported
- Improve how software written in Swift is installed and deployed
- Support **cross-platform** tooling such as an Language Server Protocol (LSP), code formatting, refactoring, and the Swift Package Manager
- Cultivate a rich open source library ecosystem

Source: https://forums.swift.org/t/on-the-road-to-swift-6/32862

Full-stack development

The Problem.

- Mobile teams are dependent on backend APIs
- Backend teams are part of big silos
- The roadmaps are not aligned

The Problem.

- General purpose backend APIs
- Meetings
- Increased costs of maintaining teams

Cross-platform and Cross-functional teams

Why?

- User stories are done faster with single smaller teams
- Less meetings
- Single focus

Why?

- Resource utilization is low
- API & ABI stability
- Clear definition of done

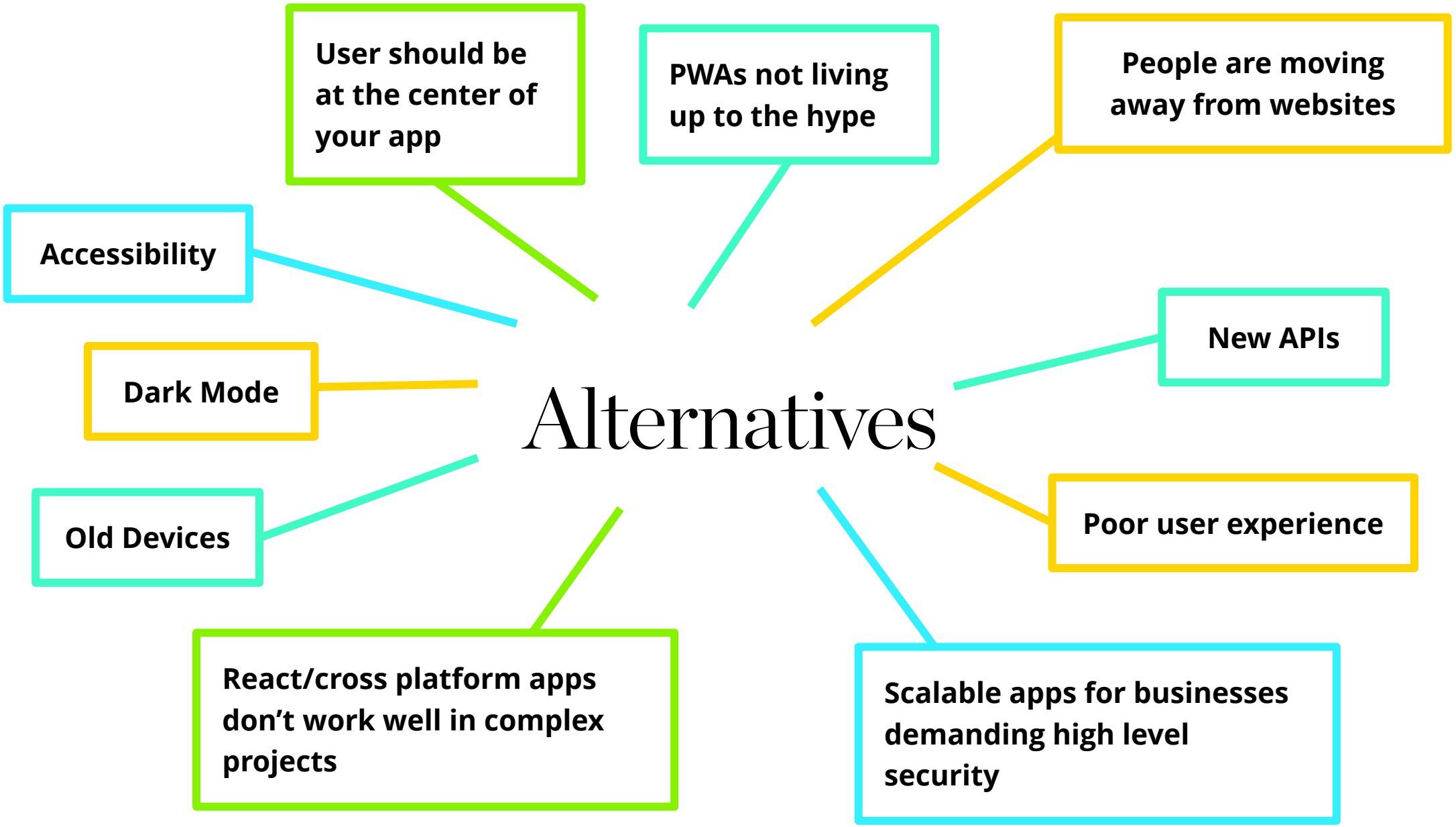
Why?

- Reuse and share models
- You have a compiler
- Debugging

User centric development

How?

- 1-3 pairs, single full-stack teams
- One full-stack developer
- One team delivers the whole cuts



Why Swift?

- Is mature enough
- Backed by Apple
- Open-source
- Modern
- Fast, close to C performance

Going Vapor

What is Vapor?

- The most popular web framework using Swift
- Works on Ubuntu and macOS
- Non-blocking, event-driven architecture built on top of Apple's SwiftNIO
- Expressive, protocol-oriented design with a focus on type-safety and maintainability
- Written in Swift for Swift developers

What is Vapor?

- Databases: SQLite, MySQL, PostgreSQL, MongoDB, Redis
- Front-end tools: Leaf
- Security: Crypto, JWT, Auth

Async

- Futures – Reference to an object that may not be available yet
- Promise – A promise to provide some object asynchronously
- Promises are used to create futures

Async

Future

.map(to:_:)

.flatMap(to:_:)

do(_:)

catch(_:)

Promise

succeed(_:)

fail(_:)

Map

- Maps a future value to a different value
- The future's value may not be available yet (it may be the result of an asynchronous task) we must provide a closure to accept the value
- Use it when you want to return a non-future value

Map

```
/// Assume we get a future string back from some API
let futureString: Future<String> = ...

/// Map the future string to an integer
let futureInt = futureString.map(to: Int.self) { string in
    print(string) // The actual String
    return Int(string) ?? 0
}

/// We now have a future integer
print(futureInt) // Future<Int>
```


Map (Vapor 4)

```
/// Assume we get a future string back from some API
let futureString: EventLoopFuture<String> = ...

/// Map the future string to an integer
let futureInt = futureString.map { string in
    print(string) // The actual String
    return Int(string) ?? 0
}

/// We now have a future integer
print(futureInt) // EventLoopFuture<Int>
```


Flat Map

- Maps a future value to different future value
- It gets the name "flat" map because it is what allows you to avoid creating nested futures (e.g., Future<Future<T>>). In other words, it helps you keep your generic futures flat
- Use it when you want to return a future value

Flat Map

```
/// Assume we get a future string back from some API
let futureString: Future<String> = ...

/// Assume we have created an HTTP client
let client: Client = ...

/// Flat-map the future string to a future response
let futureResponse = futureString.flatMap(to: Response.self) { string in
    return client.get(string) // Future<Response>
}

/// We now have a future response
print(futureResponse) // Future<Response>
```


Flat Map (Vapor 4)

```
/// Assume we get a future string back from some API
let futureString: EventLoopFuture<String> = ...

/// Assume we have created an HTTP client
let client: Client = ...

/// flatMap the future string to a future response
let futureResponse = futureString.flatMap { string in
    client.get(string) // EventLoopFuture<ClientResponse>
}

/// We now have a future response
print(futureResponse) // EventLoopFuture<ClientResponse>
```


Transform

- Maps a future to an already available value
- Allows you to modify a future's value, ignoring the existing value
- This is especially useful for transforming the results of Future<Void> where the actual value of the future is not important

Transform

```
/// Assume we get a void future back from some API
let userDidSave: Future<Void> = ...

/// Transform the void future to an HTTP status
let futureStatus = userDidSave.transform(to: HTTPStatus.ok)
print(futureStatus) // Future<HTTPStatus>
```


Transform (Vapor 4)

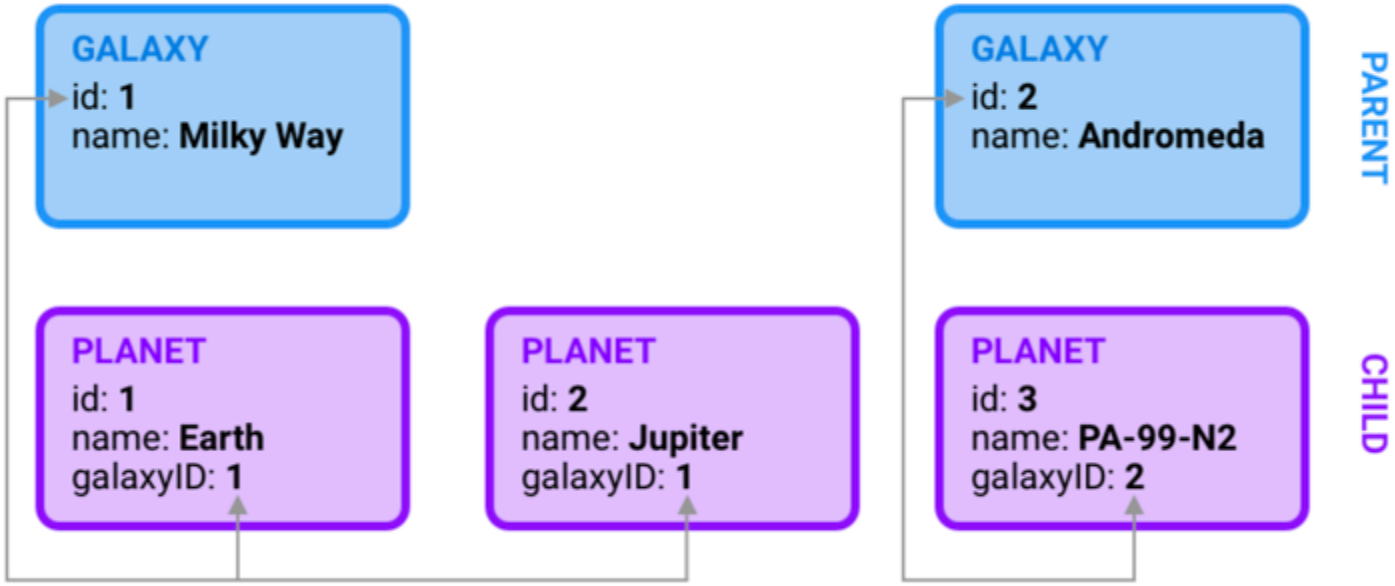
```
/// Assume we get a void future back from some API
let userDidSave: EventLoopFuture<Void> = ...

/// Transform the void future to an HTTP status
let futureStatus = userDidSave.transform(to: HTTPStatus.ok)
print(futureStatus) // EventLoopFuture<HTTPStatus>
```


Fluent

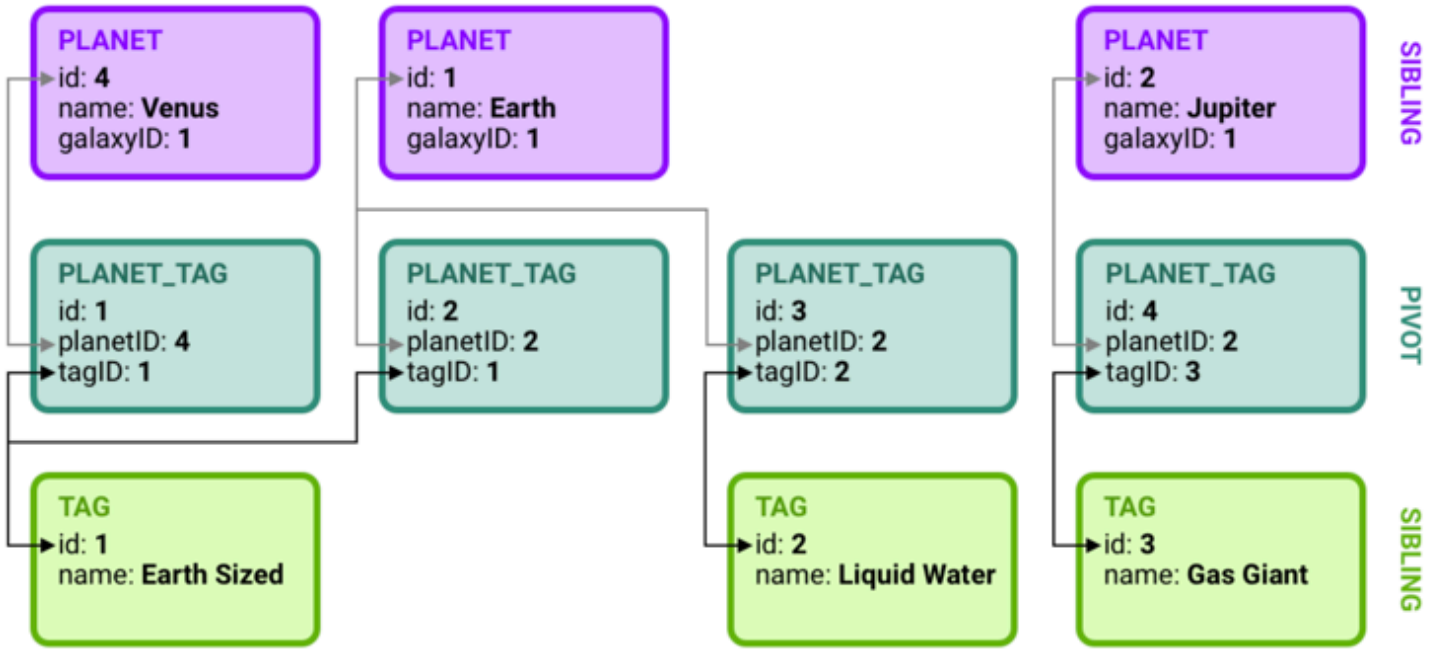
- Easy to use ORM framework built for Swift
- Support for multiple databases
- Fluent Models – used for queries

Parent-child relationships



Source: https://docs.vapor.codes/3.0/fluent/relations/

Siblings



Source: https://docs.vapor.codes/3.0/fluent/relations/

Demo

Going Vapor

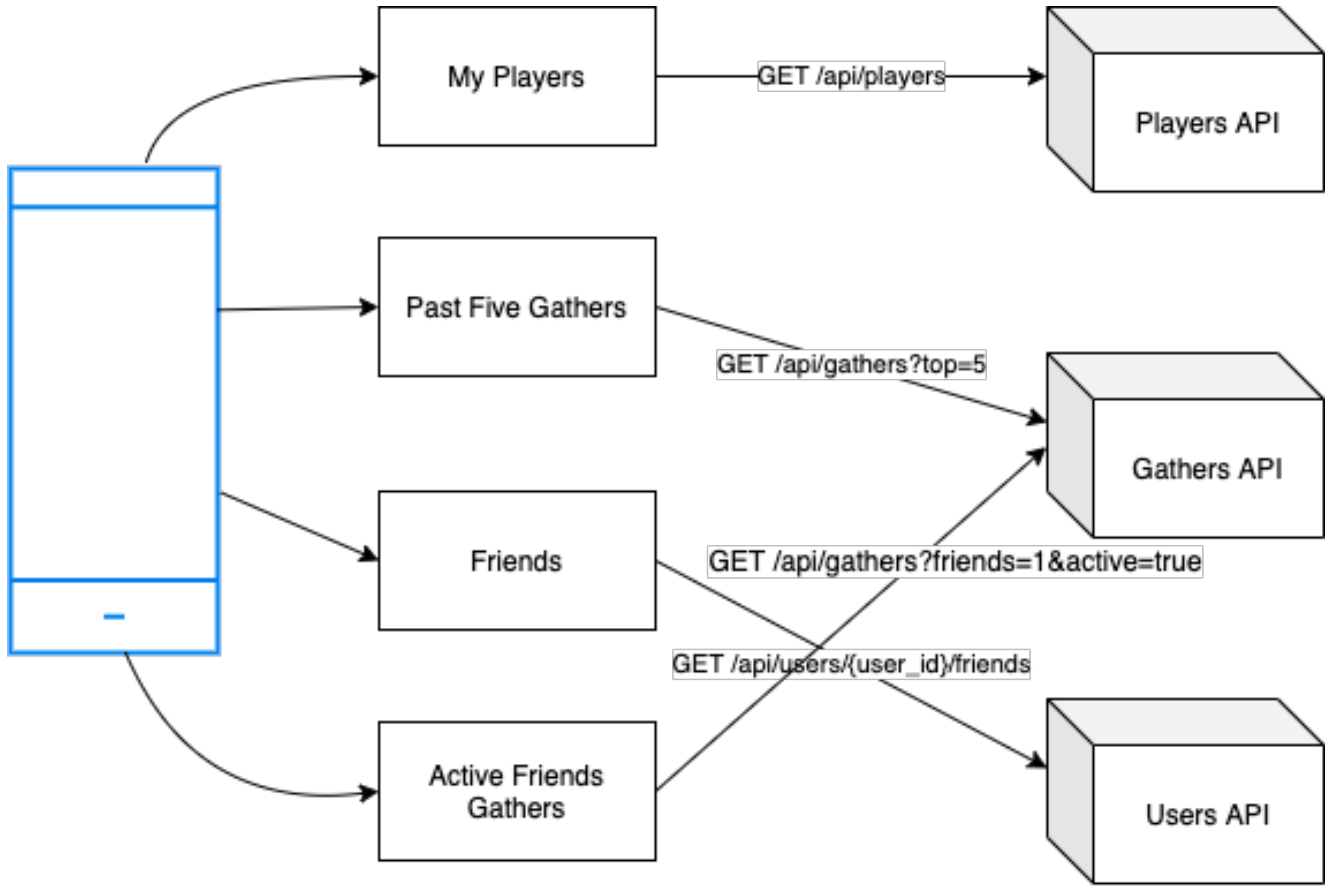
The Problem... again.

- General Purpose API serves multiple clients with a same purpose
- Different clients have different needs
- Monolith services

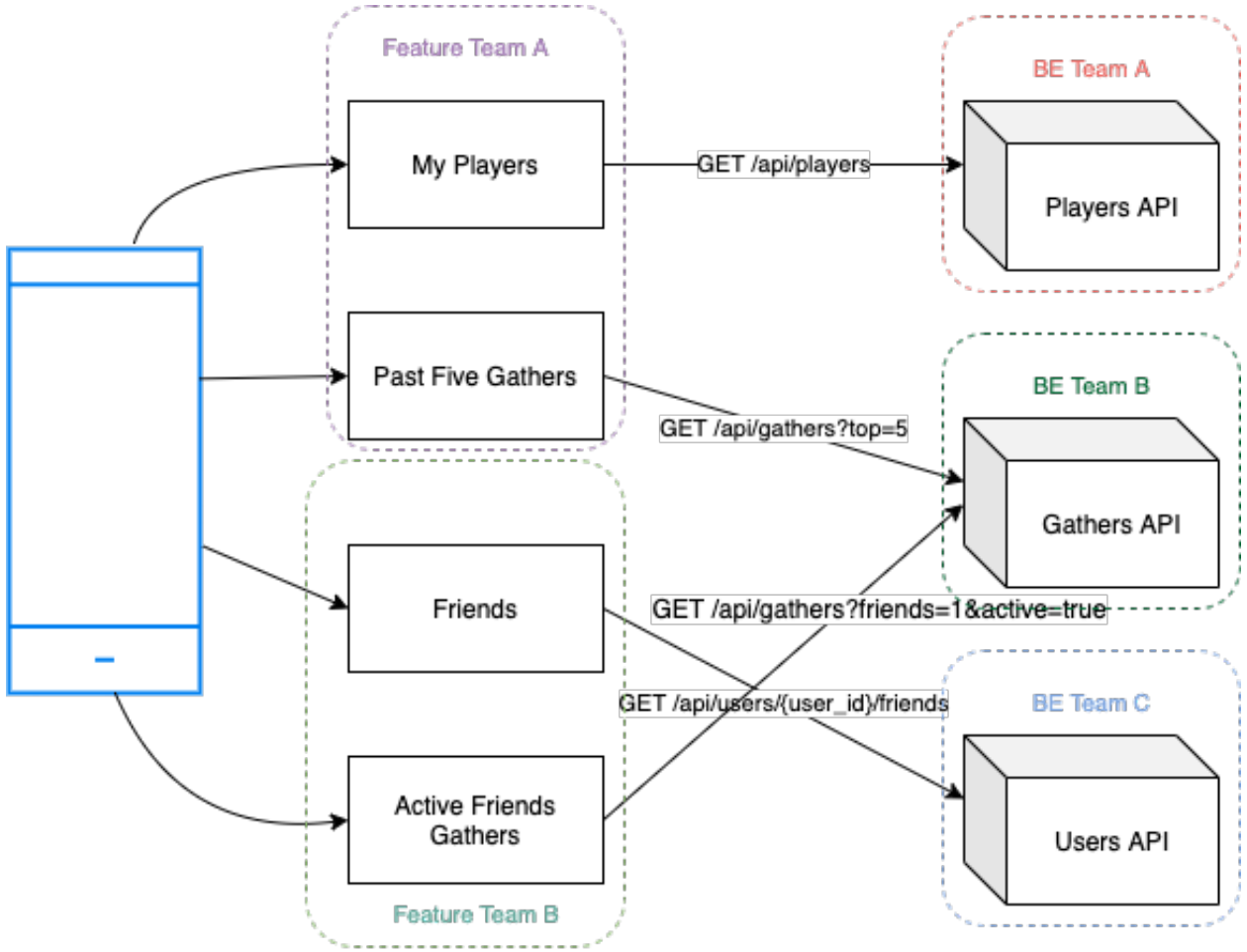
The Problem... again.

- The Home Screen problem
- Different teams manage different parts
- Performance

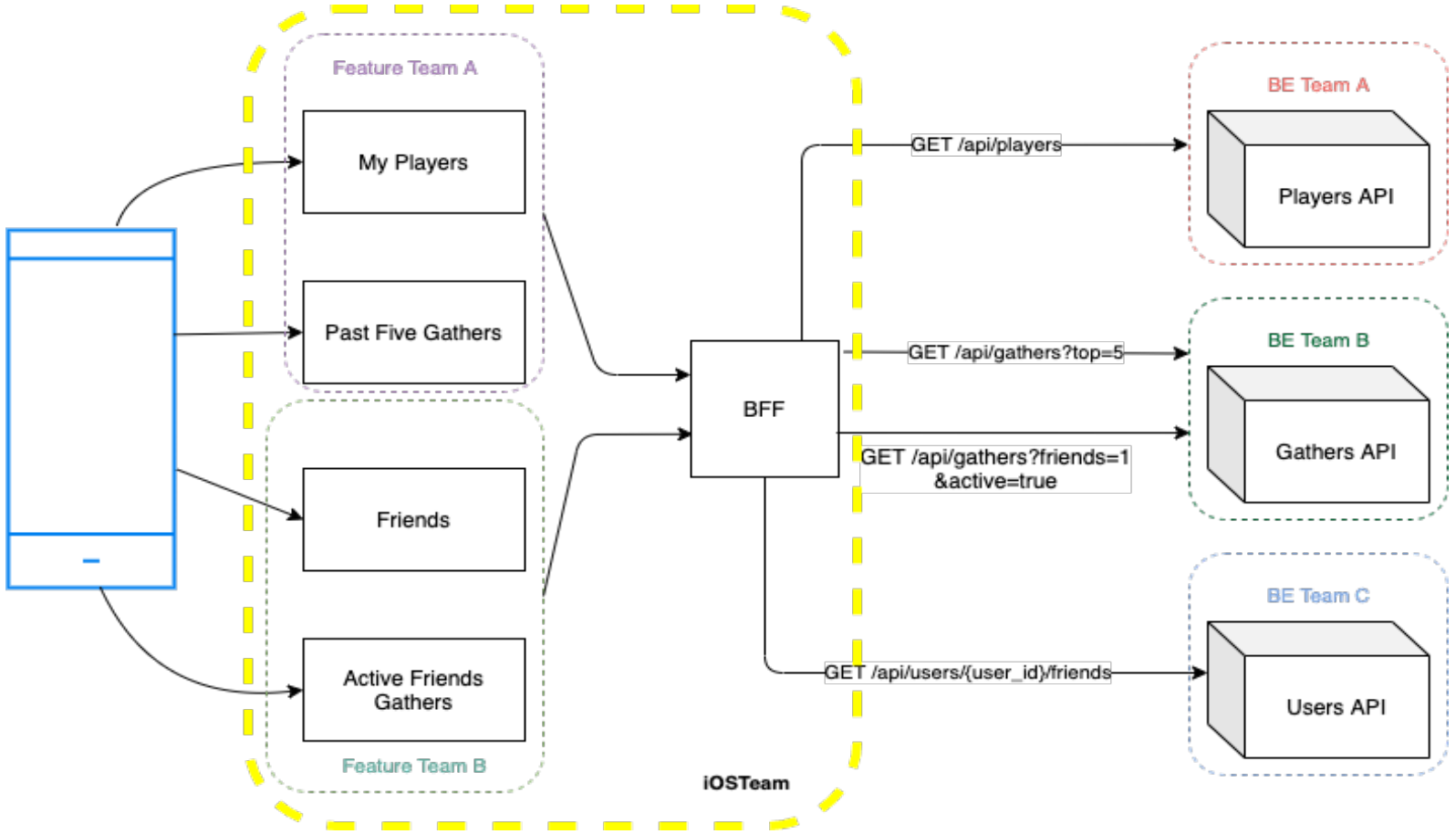
Home Screen Problem



Home Screen Problem



Introducing BFFs



Backend For Frontend

- The iOS app calls one API per screen
- No more AppStore updates
- API Team builds and maintains the APIs

Backend For Frontend

- iOS and Android teams builds and maintains the APIs
- Tightly coupled to a user experience
- Increased performance

Demo

BFFs

Why Vapor?

- Cheaper to run
- Increased developer productivity and happiness
- Increased reliability

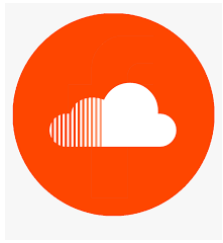
But...is it production ready?

Production Readiness

Swift NIO

Swift Server Working Group (SSWG)

Mercedes-Benz



Wait, there's more

Static Site Generator

- Requests aren’t dynamically evaluated
- Extremely scalable
- Cache friendly

1. Read the content

2. Parse the content

3. Generate HTML

Publish Overview

publish

<plot>

ink

Sweep

Codextended

files

ShellOut

Demo

Static Site Generator

Conclusion

- Server-Side Swift is cheaper to run
- Increased developer productivity
- Happier developers

Conclusion

- Safer
- Small memory footprint
- Increased reliability

Deloitte.
Digital

Thank you.

Radu Dan

Lead iOS Dev, Deloitte Digital Romania

Contact: rdan@deloittece.com

 **@radude89**

 **radude89**

 **radude89.com**