Estudo comparativo de modelos de desenvolvimento iOS nativo

Aluno: Igor Gomes Arantes

Orientador: Prof. Dr. André Backes

Banca:

Prof. Dr. Paulo Henrique Ribeiro Gabriel

Prof. Dr. Rodrigo Sanches Miani

Motivação

Do ponto de vista de desenvolvimento, diferentes abordagens de implementação se adequam melhor a certos tipos de problemas, resultando em um melhor desenvolvimento do sistema.

Impacto sentido em diferentes fatores:

- Tempo de desenvolvimento
- Escalabilidade do código
- Facilidade de produção
- etc.

Objetivo

Demonstrar que para cada situação existe uma abordagem que se adequa melhor, considerando os seguintes parâmetros:

- Simplicidade de desenvolvimento.
- Escalabilidade do código.
- Facilidade de manutenção.
- Testabilidade do sistema.

Método

Comparação de dois modelos distintos de desenvolvimento de aplicativos iOS nativos. Os seguintes tópicos foram considerados:

- Divisão de responsabilidades dentro do aplicativo.
- Comunicação com aplicações externas.
- Organização dos arquivos dentro do projeto.
- Gerenciamento de dependências externas.

Conceitos e tecnologias

Conceitos importantes

- Aplicativo nativo
- *View*
- *ViewController*

Tecnologias utilizadas

- XCode
- Swift
- *The Movie Database*

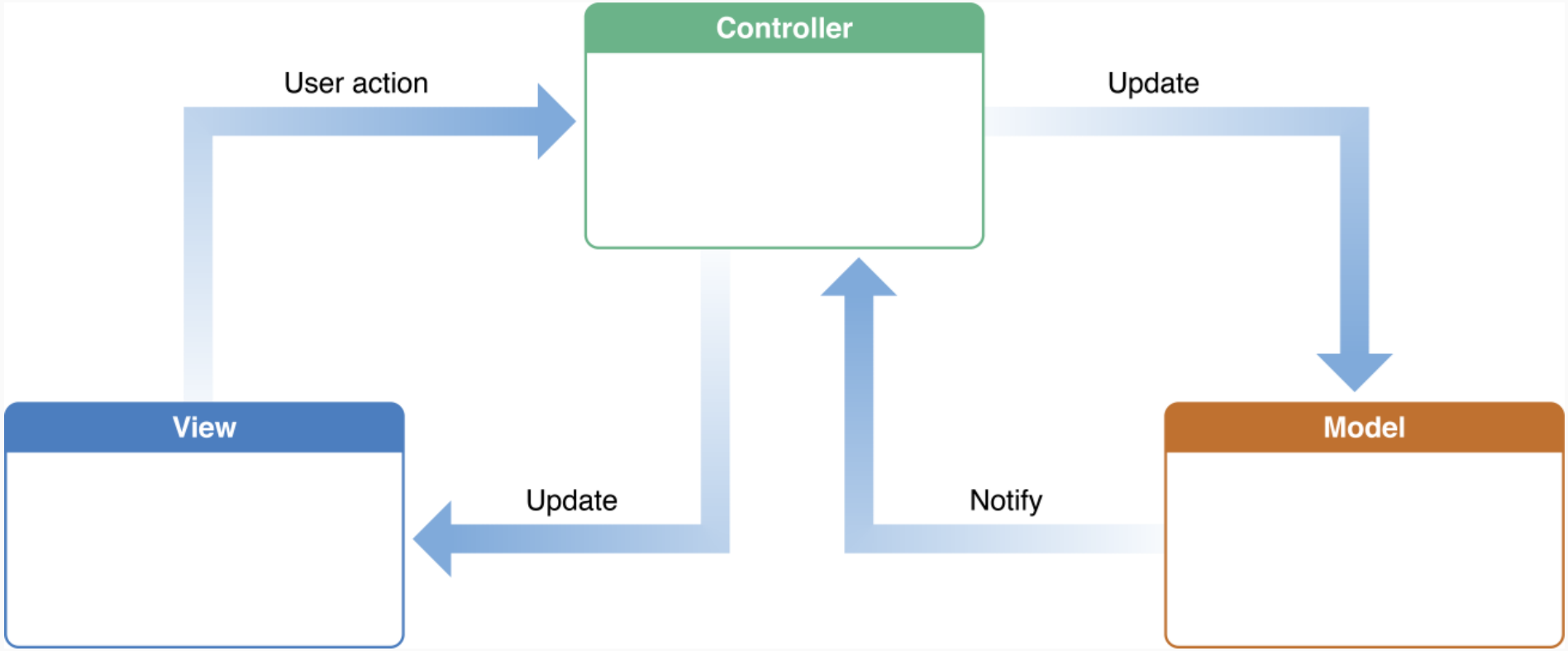
Modelos implementados

	Modelo A	**Modelo B**
Padrão arquitetural	MVVM-C	MVC
Camada de serviços	Modularizada	Centralizada
Interface visual	View code	Storyboard
Organização de arquivos	Contextual	Funcional
Gerenciador de dependências	Carthage	CocoaPods

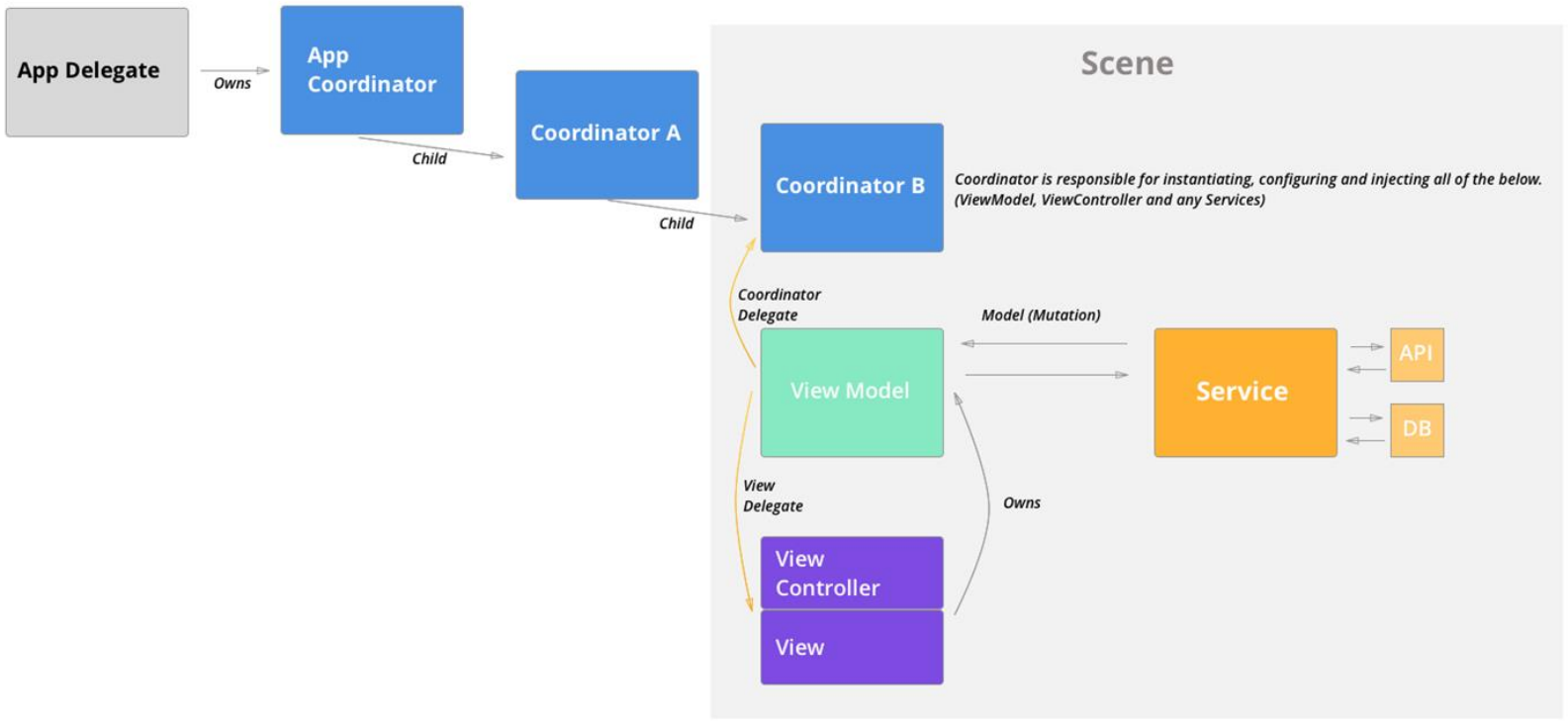
Divisão de responsabilidades

	Modelo A: MVVM-C	**Modelo B: MVC**
Navegação	Coordinator	ViewController
Lógica de negócios	ViewModel	ViewController
Controle	ViewController	ViewController
Interface visual	View	View
Modelo de dados	Model	Model

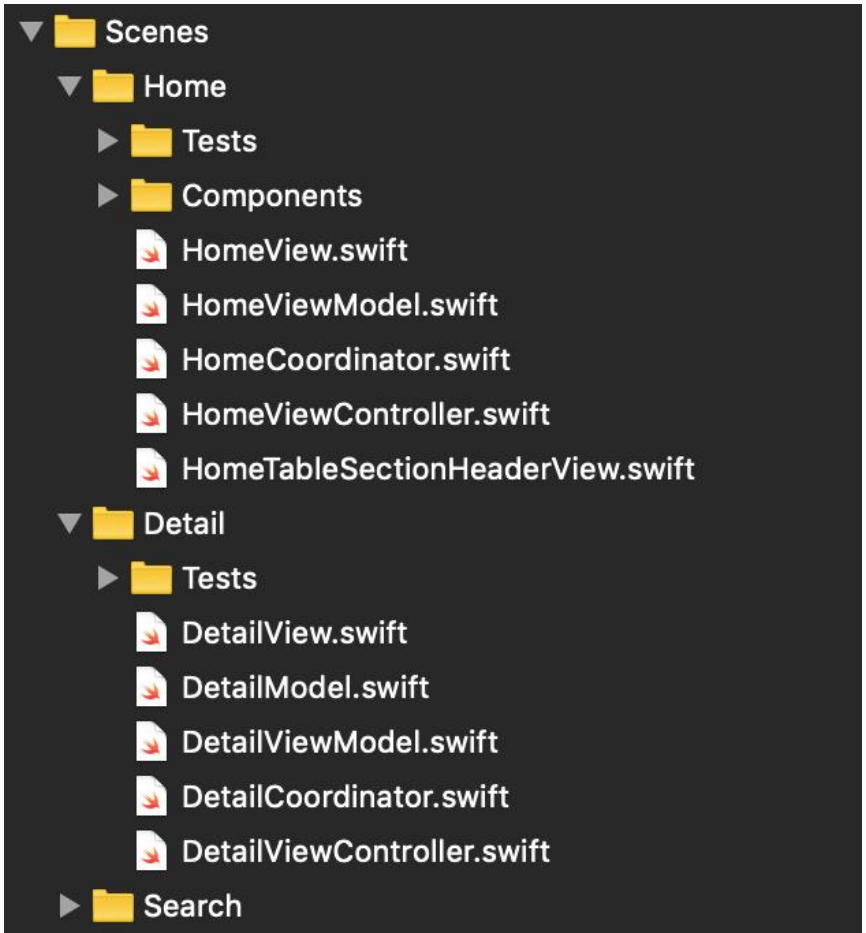
MVC



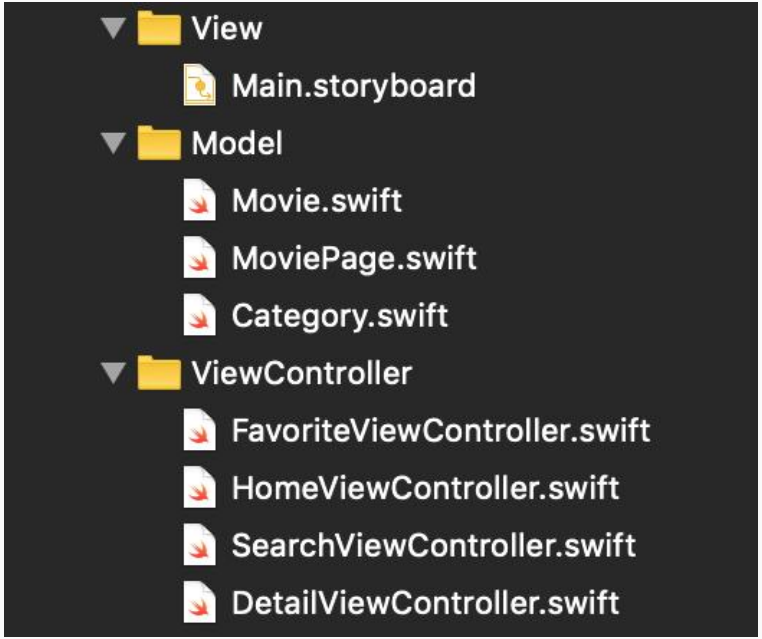
MVVM-C


App Delegate
Owns
App Coordinator
Child
Coordinator A
Child
Scene
Coordinator B
Coordinator is responsible for instantiating, configuring and injecting all of the below. (ViewModel, ViewController and any Services)
Coordinator Delegate
Model (Mutation)
View Model
Service
API
DB
View Delegate
Owns
View Controller
View

Modelo A - Organização Contextual



Modelo B - Organização Funcional



Modelo A - Camada de serviços modularizada

```
protocol EndPointProtocol {
    var baseURL: URL { get }
    var path: String { get }
    var mockLocalPath: String { get }
    var headers: HTTPHeaders? { get }
    var httpMethod: HTTPMethod { get }
    var httpParameters: HTTPParameters { get }
}
```

Modelo B - Camada de serviços centralizada

```
protocol MovieServiceProtocol {
    func getMovieDetail(id: Int, completion: @escaping (Response<Movie>) -> ())
    func getMoviePage(page: Int, sort: Sort, order: Order, completion: @escaping (Response<MoviePage>) -> ())
    func getMoviePageByName(query: String, completion: @escaping (Response<MoviePage>) -> ())
    func getTrendingMoviePage(page: Int, completion: @escaping (Response<MoviePage>) -> ())
}
```

Gerenciamento de dependências

	Modelo A: Carthage	**Modelo B: CocoaPods**
Repositório de bibliotecas	Descentralizado	Centralizado
Dificuldade de uso	Fácil	Muito fácil
Controle da estrutura do projeto	Alto	Baixo

Conclusão

	Modelo A	Modelo B
Escalabilidade do código	Alta	Baixa
Facilidade de manutenção	Alta	Média
Reusabilidade de componentes	Alta	Baixa
Simplicidade no desenvolvimento	Média	Alta
Testabilidade do sistema	Alta	Baixa