etermax

Arquitectura y Modularización en Preguntados

SwiftBA

Pablo Manuelli
Technical Owner @Preguntados
@pablomanuelli

Lucas David Serruya
Technical Owner @Preguntados
@ldserruya

etermax

Agenda

- Interaction Driven Design
  - Arquitectura basada en IDD
  - Ventajas
- Modularización

etermax

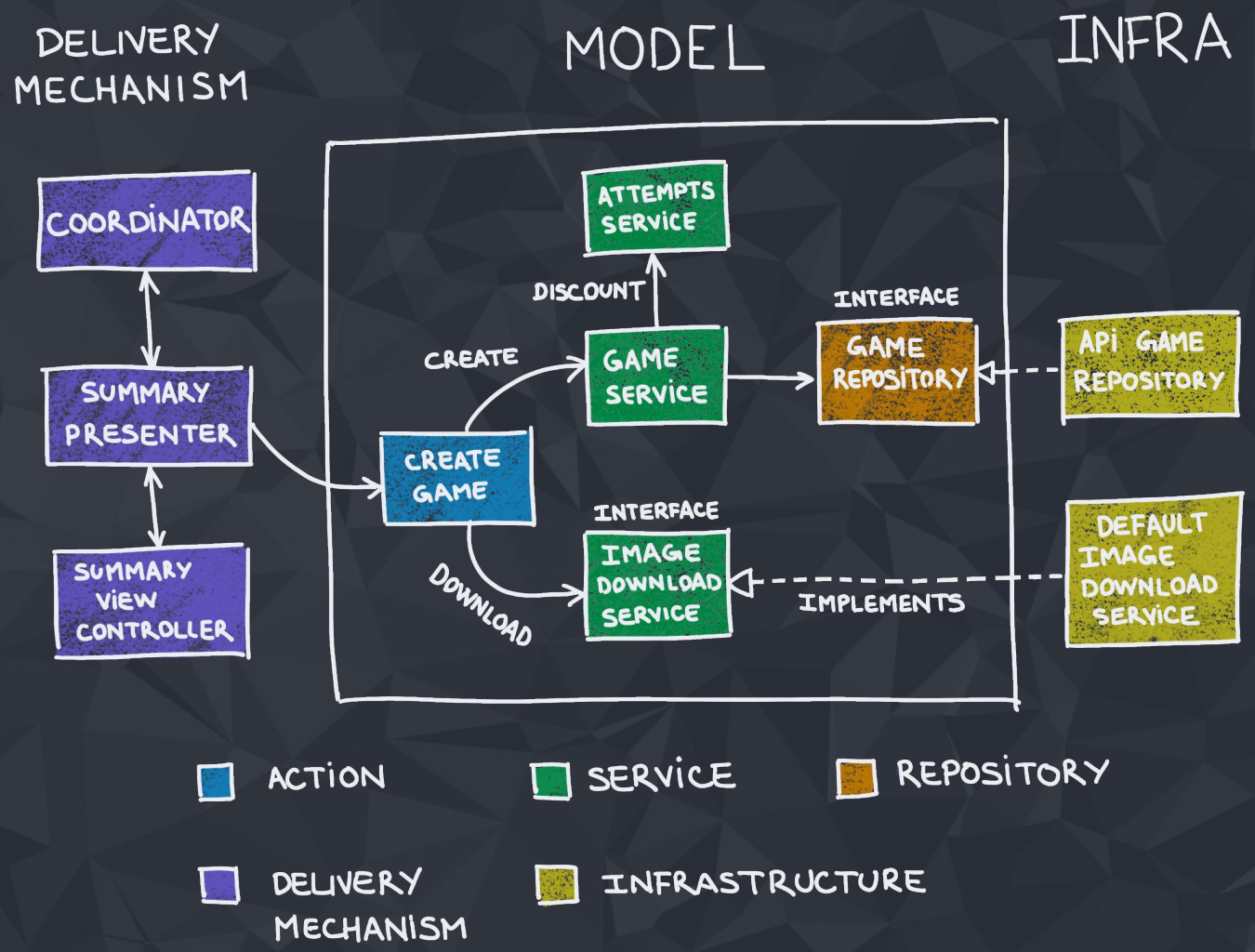
Interaction Driven Design

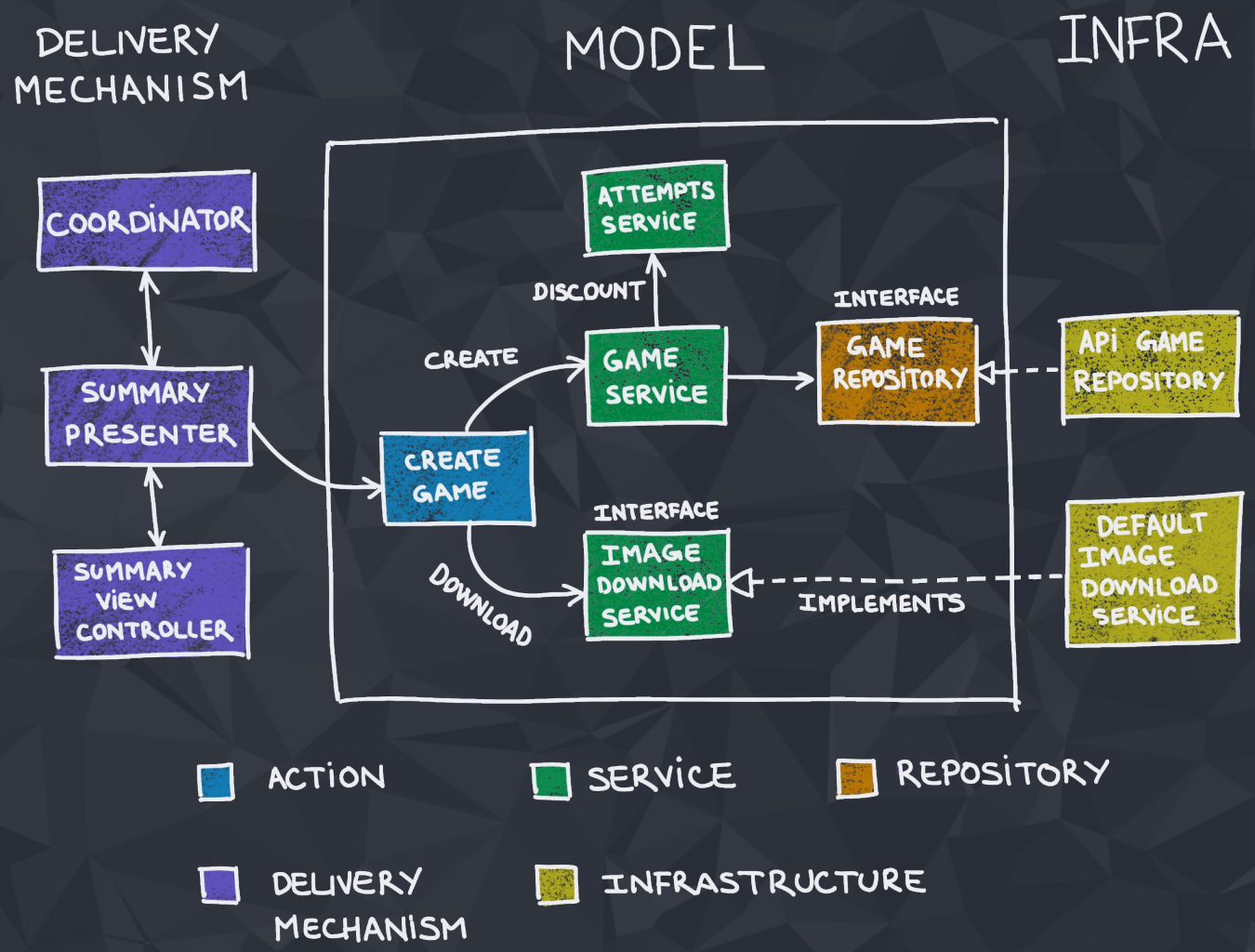
¿Qué es IDD?

- Metodología de diseño y desarrollo de software basado en Outside-In Development.
- Se hace foco en el uso externo del sistema.
- Desarrollo iterativo.

etermax

Arquitectura de un módulo de Preguntados basada en IDD


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
INTERFACE
CREATE
GAME SERVICE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

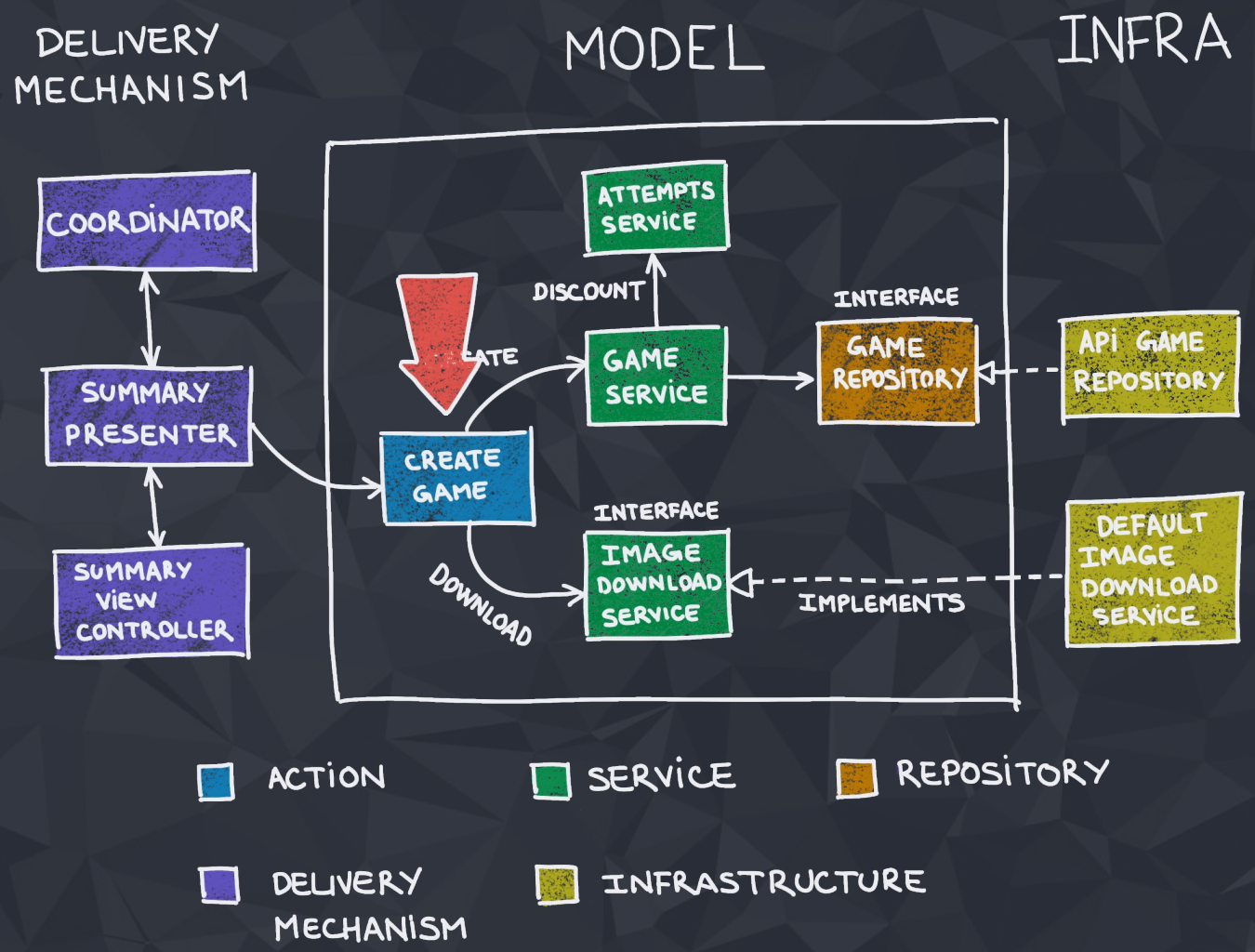

DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
INTERFACE
CREATE
GAME SERVICE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

etermax

Componentes de Arquitectura

Actions

- Representan acciones que pueden realizar los usuarios para interactuar con el sistema.
- Son el punto de entrada al modelo de la aplicación o del feature.
- Exponen un único método público llamado *execute*.
- Son ejecuciones a alto nivel, delegan los detalles a los servicios.


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
INTERFACE
GAME SERVICE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

etermax

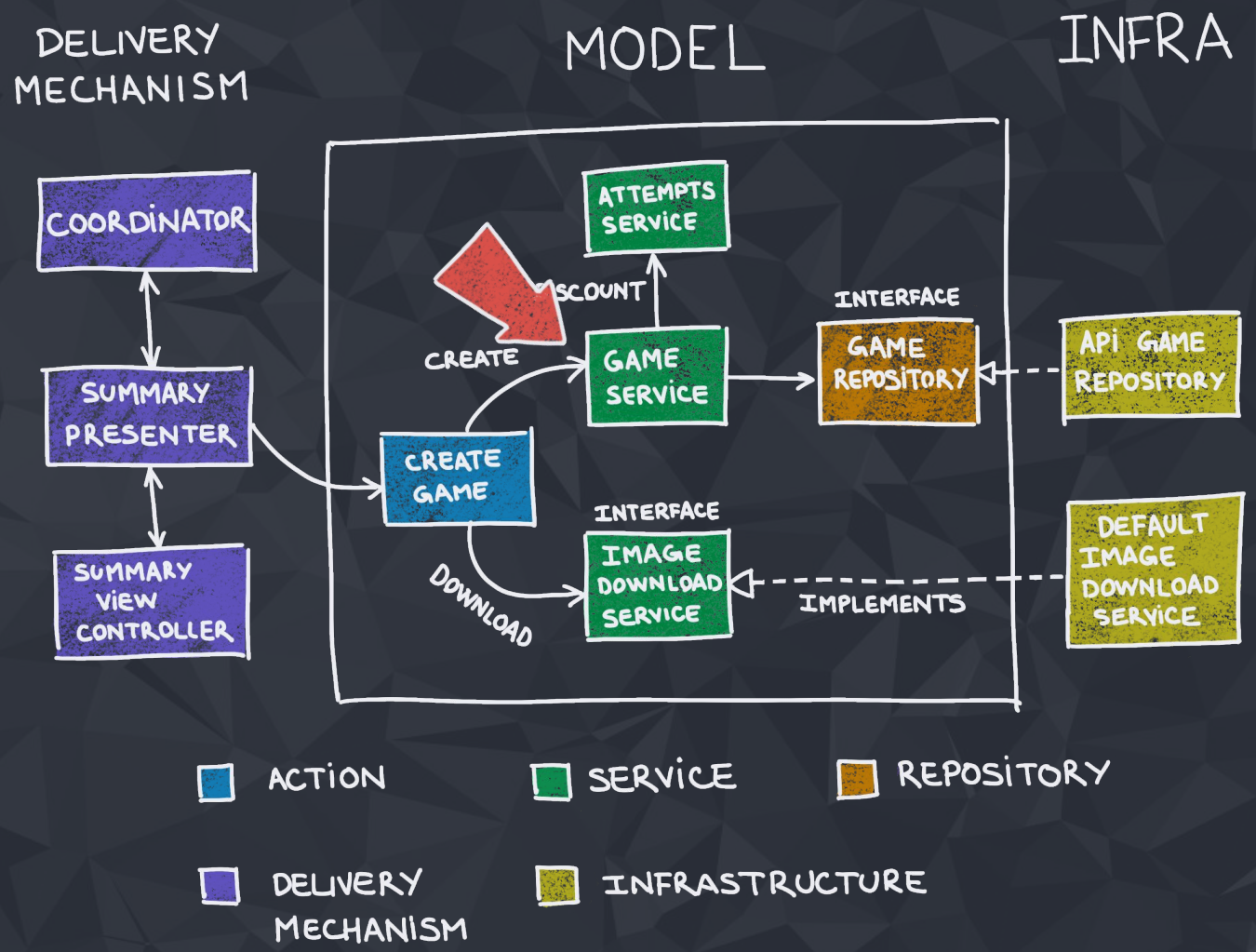
```
protocol CreateGame {

    func execute(topic: Topic) -> Single<Game>
}

class DefaultCreateGame: CreateGame {

    private let gameService: GameService
    private let imageDownloadService: ImageDownloadService

    init(gameService: GameService, imageDownloadService: ImageDownloadService) {
        self.gameService = gameService
        self.imageDownloadService = imageDownloadService
    }

    func execute(topic: Topic) -> Single<Game> {
        return gameService
            .createGame(topic: topic)
            .do(onSuccess: { game in self.downloadImages(in: game) })
    }

    private func downloadImages(in game: Game) {
        imageDownloadService.downloadImages(in: game.questions)
    }
}
```

etermax

```
protocol CreateGame {

    func execute(topic: Topic) -> Single<Game>
}

class DefaultCreateGame: CreateGame {

    private let gameService: GameService
    private let imageDownloadService: ImageDownloadService

    init(gameService: GameService, imageDownloadService: ImageDownloadService) {
        self.gameService = gameService
        self.imageDownloadService = imageDownloadService
    }

    func execute(topic: Topic) -> Single<Game> {
        return gameService
            .createGame(topic: topic)
            .do(onSuccess: { game in self.downloadImages(in: game) })
    }

    private func downloadImages(in game: Game) {
        imageDownloadService.downloadImages(in: game.questions)
    }
}
```

Services

- Contienen a la mayoría de lógica de dominio.
- Comportamiento relacionado a múltiples instancias de la misma entidad o diferentes entidades.
- Comportamiento que no aplica específicamente a ninguna entidad.


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
CREATE
GAME SERVICE
INTERFACE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

```swift
protocol GameService {

    func createGame(topic: Topic) -> Single<Game>
}

class DefaultGameService: GameService {

    private let attemptsService: AttemptsService
    private let gameRepository: GameRepository

    init(attemptsService: AttemptsService, gameRepository: GameRepository) {
        self.attemptsService = attemptsService
        self.gameRepository = gameRepository
    }

    func createGame(topic: Topic) -> Single<Game> {
        guard attemtsService.hasAvailableAttempts(for: topic) else {
            return .error(.notAvailableAttempts)
        }

        return gameRepository
            .create(topic: topic)
            .do(onSuccess: { _ in self.attemptsService.discountAttempt(for: topic) })
    }
}
```

Entities y Value Objects

- Representan elementos del modelo.
- Las Entities representan elementos del modelo cuya identidad es importante. Sus valores pueden mutar con el tiempo.
- Los value objects por el contrario son elementos cuya identidad no es relevante y son inmutables.

```swift
class Game {

    let id: GameId
    let topic: Topic
    let questions: [Question]

    private(set) var score: Score
    private(set) var isFinished = false

    init(id: GameId, topic: Topic, questions: [Question], score: Score) {
        self.id = id
        self.topic = topic
        self.questions = questions
        self.score = score
    }

    func answerQuestion(with answer: Answer) {

        if answer.isCorrect {
            score = score.increatedByOne()
        }
        else {
            isFinished = true
        }
    }
}
```

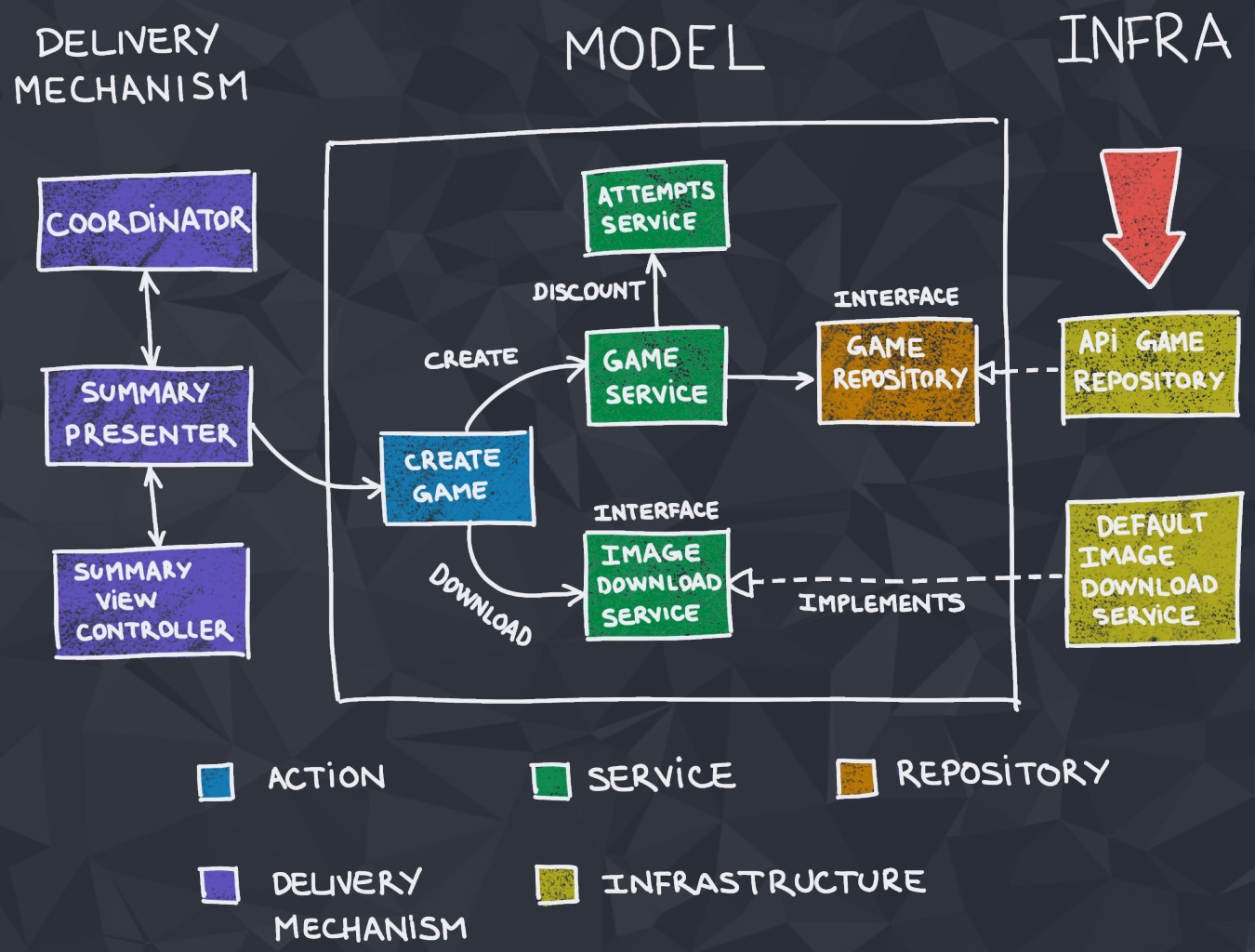
```swift
struct Score {

    let value: Int

    func increaseByOne() -> Score {
        return Score(value: value + 1)
    }
}
```

```swift
class Game {

    let id: GameId
    let topic: Topic
    let questions: [Question]

    private(set) var score: Score
    private(set) var isFinished = false

    init(id: GameId, topic: Topic, questions: [Question], score: Score) {
        self.id = id
        self.topic = topic
        self.questions = questions
        self.score = score
    }

    func answerQuestion(with answer: Answer) {

        if answer.isCorrect {
            score = score.increatedByOne()
        }
        else {
            isFinished = true
        }
    }
}
```

```swift
struct Score {

    let value: Int

    func increaseByOne() -> Score {
        return Score(value: value + 1)
    }
}
```

Repositories

- Representan colecciones de objetos del dominio.
- Su protocolo pertenece al modelo pero su implementación a la infraestructura.
- Devuelve objetos de dominio.
- Los objetos de dominio no implementan Decodable.


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
CREATE
GAME SERVICE
INTERFACE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

etermax

```
protocol GameRepository {

    func create(topic: Topic) -> Single<Game>
}

class ApiGameRepository: GameRepository {

    func create(topic: Topic) -> Single<Game> {

        let request = buildCreateGameRequest(topic: topic)

        return URLSession.shared.rx
            .data(request: request).asSingle()
            .map { data in self.parseGameData(data) }
    }

    private func parseGameData(_ data: Data) -> Game {
        // Use a Decodable to parse the data,
        // then create a Game from the Decodable
    }
}
```

etermax

```
struct GameDTO: Decodable {

    let id: String
    let topic: TopicDTO
    let questions: [QuestionDTO]
    let score: Int

    func toGame() -> Game {
        // Mapping code
    }
}
```

```
struct TopicDTO: Decodable {

    // Decodable properties

    func toTopic() -> Topic {
        // Mapping code
    }
}

struct QuestionDTO: Decodable {

    // Decodable properties

    func toQuestion() -> Question {
        // Mapping code
    }
}
```

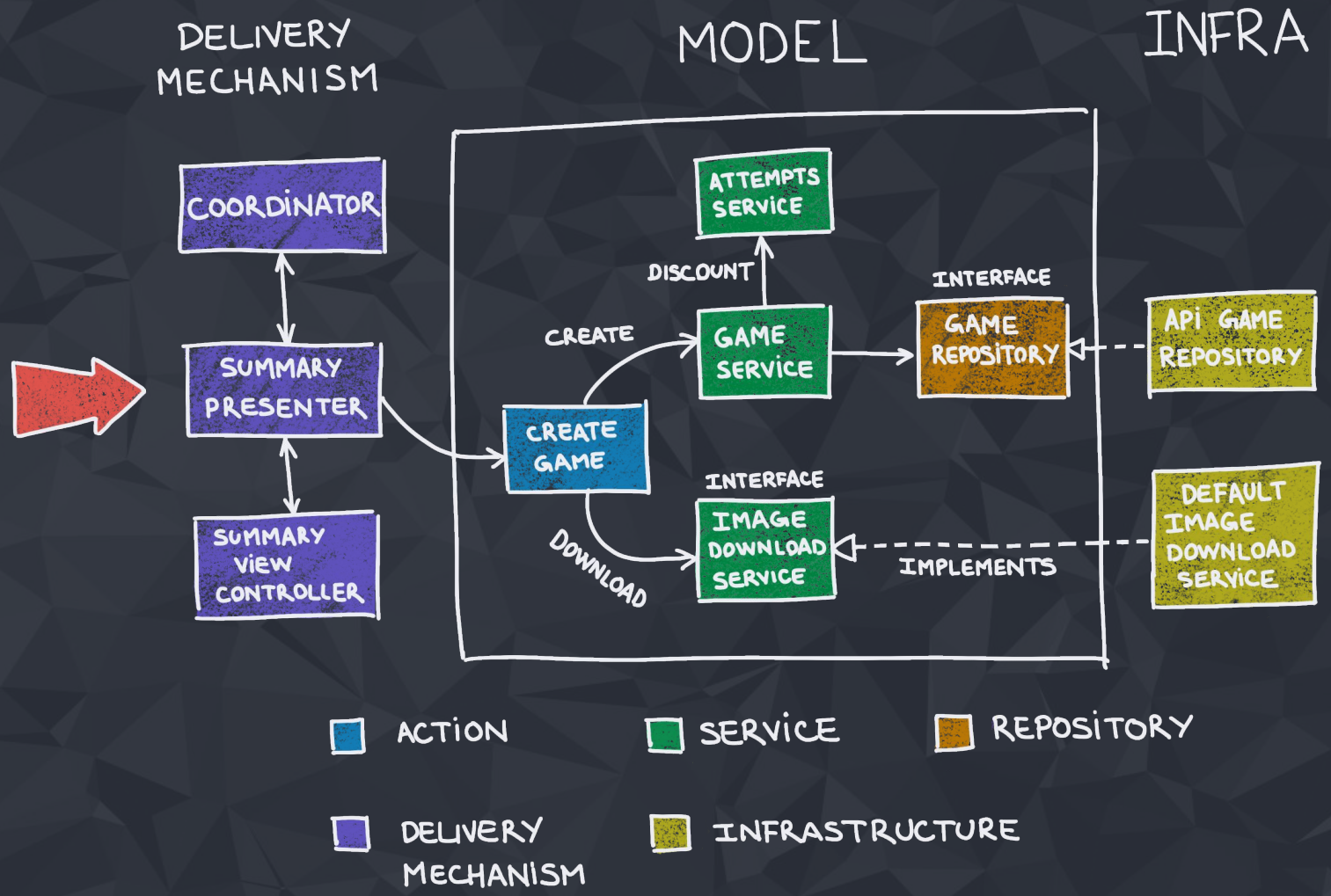
etermax

```swift
struct GameDTO: Decodable {

    let id: String
    let topic: TopicDTO
    let questions: [QuestionDTO]
    let score: Int

    func toGame() -> Game {
        // Mapping code
    }
}
```

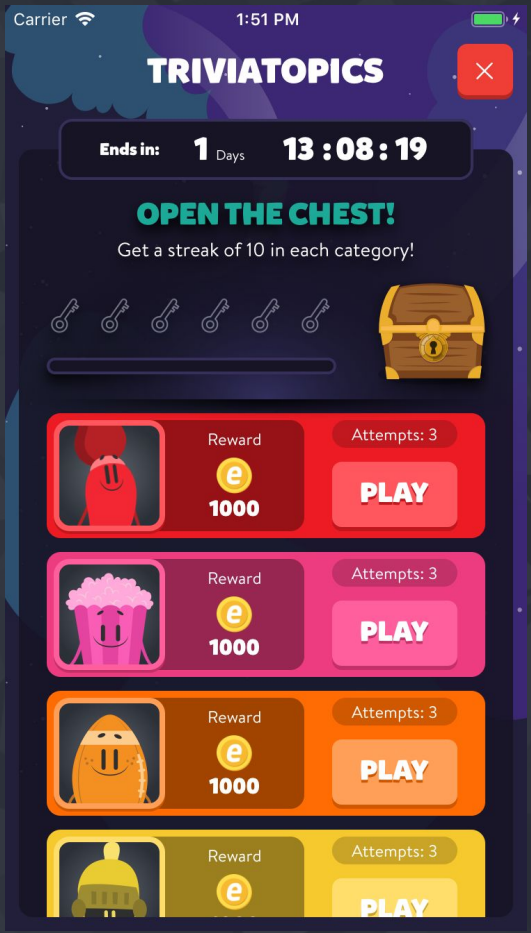
```swift
class Game {

    let id: GameId
    let topic: Topic
    let questions: [Question]

    private(set) var score: Score
    private(set) var isFinished = false

    init(id: GameId, topic: Topic, questions: [Question], score: Score) {
        self.id = id
        self.topic = topic
        self.questions = questions
        self.score = score
    }

    func answerQuestion(with answer: Answer) {

        if answer.isCorrect {
            score = score.increatedByOne()
        }
        else {
            isFinished = true
        }
    }
}
```

Presenters

- Pertenecen al mecanismo de delivery.
- Representa al Presenter en un esquema MVP.
- Posee lógica de presentación, independiente de UIKit.
- Interactúa con las acciones del modelo.
- Interactúa con el Coordinator.


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
INTERFACE
CREATE
GAME SERVICE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

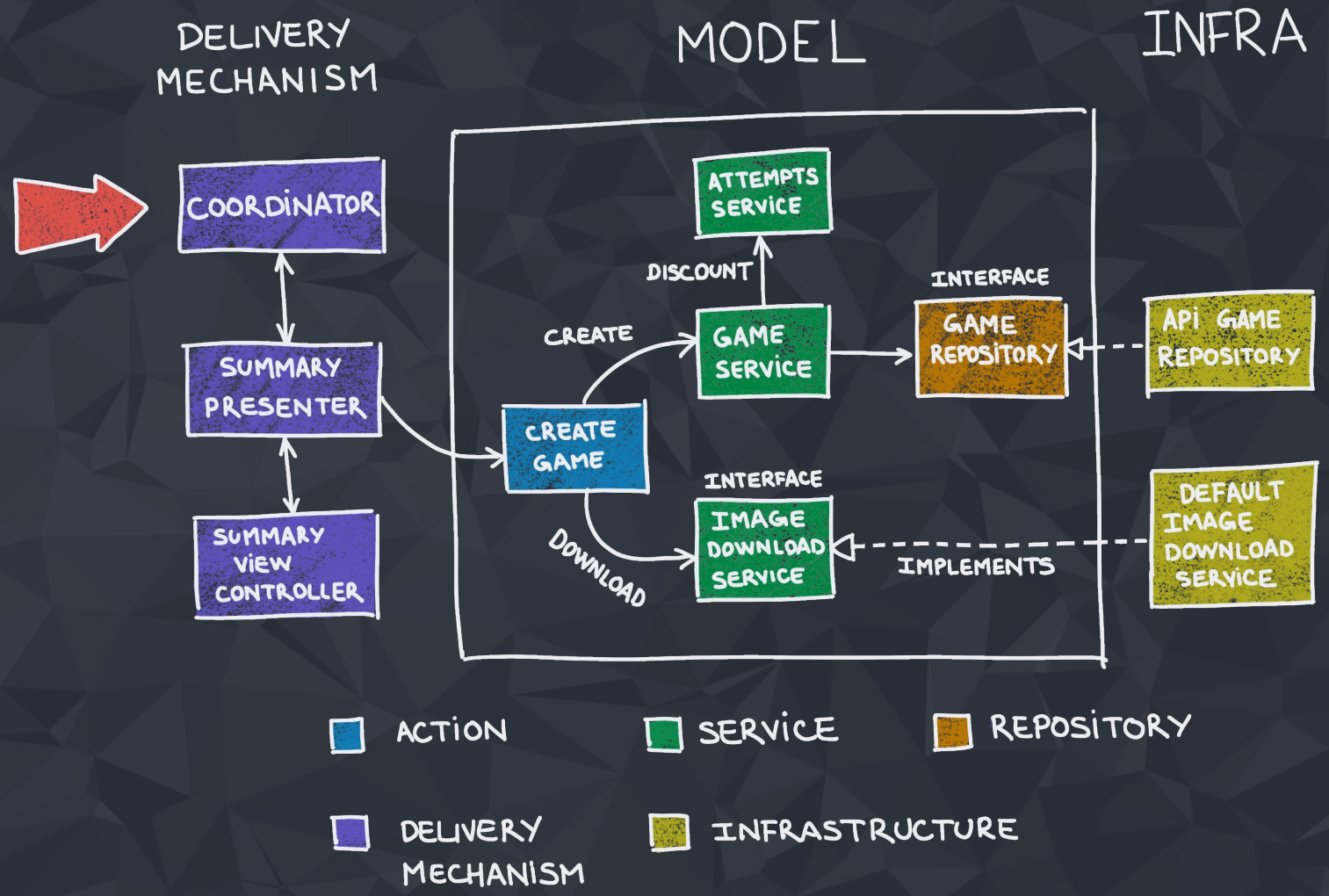
etermax


Carrier
1:51 PM
TRIVIATOPICS
Ends in:
1 Days
13 : 08 : 19
OPEN THE CHEST!
Get a streak of 10 in each category!
Reward
1000
Attempts: 3
PLAY
Reward
1000
Attempts: 3
PLAY
Reward
1000
Attempts: 3
PLAY
Reward
Attempts: 3

```swift
protocol SummaryPresenter {

    func setView(_ view: SummaryView)
    func topicSelected(at index: Int)
}

class DefaultSummaryPresenter: SummaryPresenter {

    private let coordinator: SummaryCoordinator
    private let createGame: CreateGame

    private weak var view: SummaryView?

    init(coordinator: SummaryCoordinator, createGame: CreateGame) {
        self.coordinator = coordinator
        self.createGame = createGame
    }

    func setView(_ view: SummaryView) { self.view = view }

    func topicSelected(at index: Int) {

        createGame
            .execute(getTopic(at: index))
            .subscribe(onSuccess: { game in self.coordinator.gameCreated(game) })
    }
}
```

Coordinators

- Pertenecen al mecanismo de delivery.
- Es el punto de entrada al módulo.
- Se encarga de la navegación entre distintas pantallas.
- Se pueden utilizar otros coordinators dentro del módulo si los flujos son muy complejos.


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
INTERFACE
CREATE
GAME SERVICE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

etermax

```
class Coordinator {

    private let navigationController: UINavigationController

    init(navigationController: UINavigationController) {
        self.navigationController = navigationController
    }

    func start() {
        showSummaryController()
    }
}
```

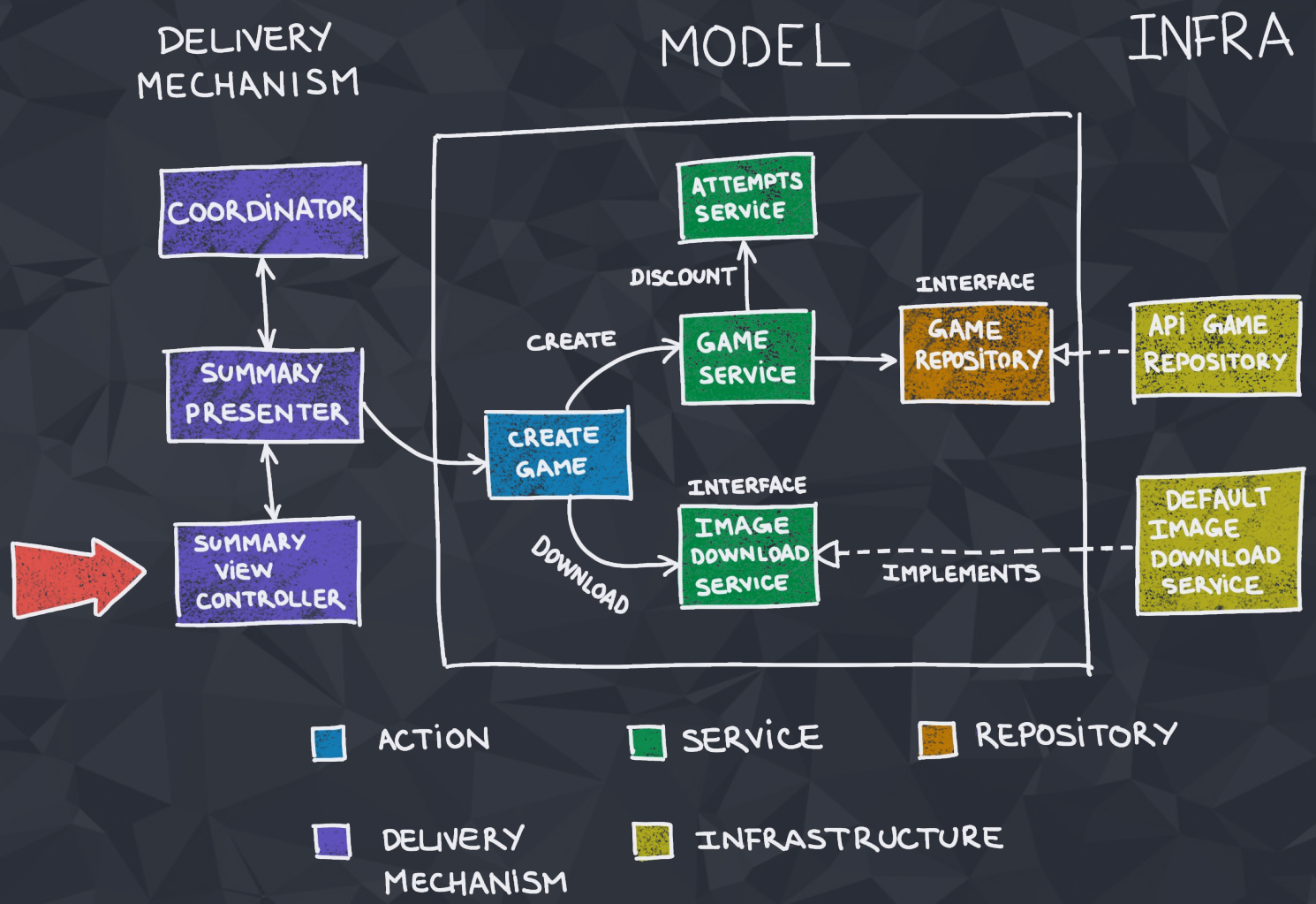
```
protocol SummaryCoordinator {

    func gameCreated(_ game: Game)
}

extension Coordinator: SummaryCoordinator {

    private func showSummaryController() {

        let createGame = CreateGameFactory.createGame()
        let presenter = DefaultSummaryPresenter(coordinator: self, createGame: createGame)
        let viewController = SummaryViewController(presenter: presenter)

        navigationController.pushViewController(viewController, animated: true)
    }

    func gameCreated(_ game: Game) {
        showGameController(game)
    }
}
```

etermax

```
class CreateGameFactory {

    static func createGame() -> CreateGame {

        let gameRepository = ApiGameRepository()
        let attemptsService = DefaultAttemptsService()
        let imageDownloadService = DefaultImageDownloadService()

        let gameService = DefaultGameService(attemptsService: attemptsService,
                                             gameRepository: gameRepository)

        return DefaultCreateGame(gameService: gameService,
                                 imageDownloadService: imageDownloadService)
    }
}
```

View Controllers

- Pertenecen al mecanismo de delivery.
- Representa a la vista en un esquema MVP o MVVM.
- Es la vista propiamente dicha y depende de UIKit.
- No tienen lógica de navegación.
- No suelen ser testeados.


DELIVERY MECHANISM
MODEL
INFRA
COORDINATOR
SUMMARY PRESENTER
SUMMARY VIEW CONTROLLER
ATTEMPTS SERVICE
DISCOUNT
INTERFACE
CREATE
GAME SERVICE
GAME REPOSITORY
API GAME REPOSITORY
CREATE GAME
INTERFACE
IMAGE DOWNLOAD SERVICE
DOWNLOAD
IMPLEMENTS
DEFAULT IMAGE DOWNLOAD SERVICE
ACTION
SERVICE
REPOSITORY
DELIVERY MECHANISM
INFRASTRUCTURE

etermax

```
protocol SummaryView: class {

    func setTimeLeftText(_ text: String)
}

class SummaryViewController: UIViewController, SummaryView {

    @IBOutlet var timeLeftLabel: UILabel!

    private let presenter: SummaryPresenter

    init(presenter: SummaryPresenter) {
        self.presenter = presenter
        super.init(nibName: "SummaryViewController", bundle: nil)
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        presenter.setView(self)
    }

    func setTimeLeftText(_ text: String) {
        timeLeftLabel.text = text
    }
}

extension SummaryViewController: UITableViewDelegate {

    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        presenter.topicSelected(at: indexPath.row)
    }
}
```

etermax

Ventajas de arquitectura IDD

- Orientada a casos de uso.
- Arquitectura separada en capas.
- Componentes testeables.
- Permite diferir decisiones de infraestructura.
- Ayuda al desarrollo Full Stack

etermax

Modularización

Contexto: Equipo Preguntados

- Aplicación utilizada por millones de usuarios diarios.
- Equipo de 60 personas, conformado por 6 squads con diferentes focos de negocio.
- Necesidad de introducir cambios con un mínimo riesgo.

Ventajas de utilizar módulos

- Facilita trabajo en simultáneo en la aplicación por varias personas.
- Hace mucho más visibles las dependencias externas.
- Pueden ser compilados y testeados individualmente.
- Facilita el versionado de features.

Referencias

- Introducing Interaction Driven Design
- Interaction Driven Design (slides)
- A Case for Outside-In Development

etermax

¿Preguntas?

etermax

¡GRACIAS!