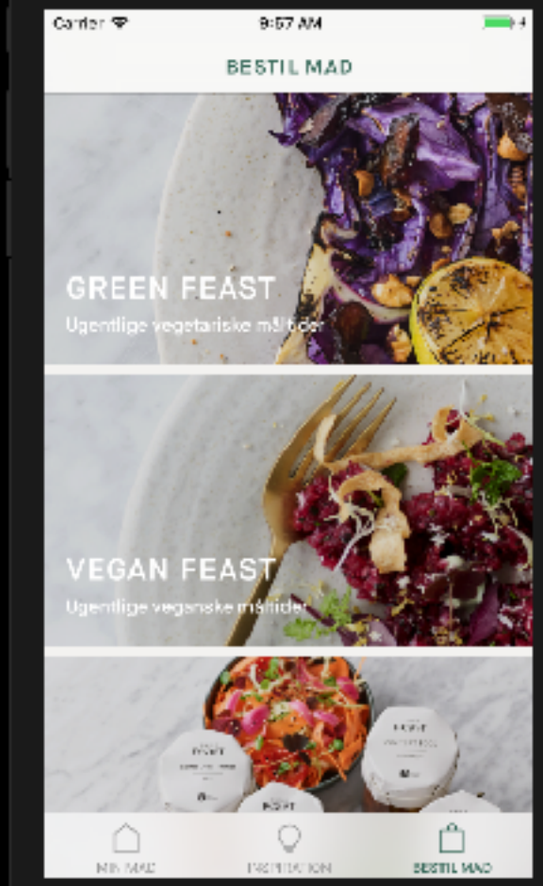
MVVM and Coordinators

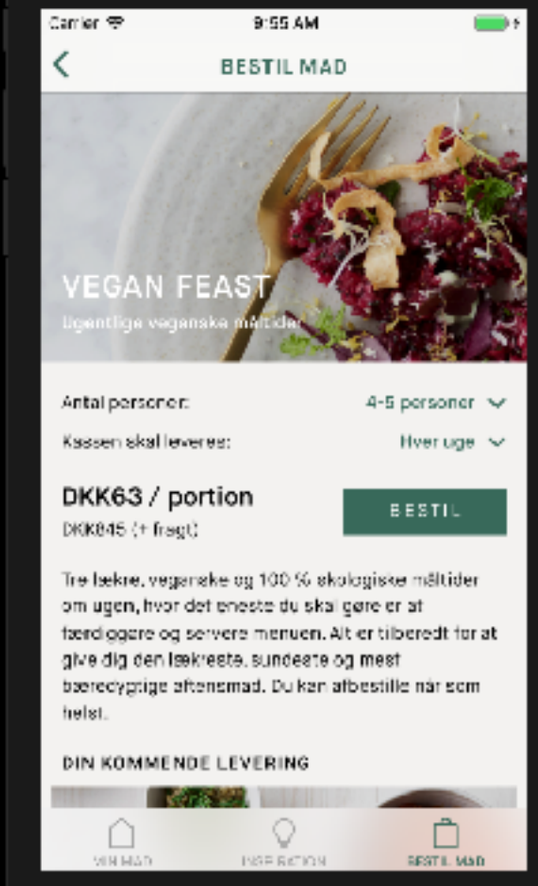
SIMPLE

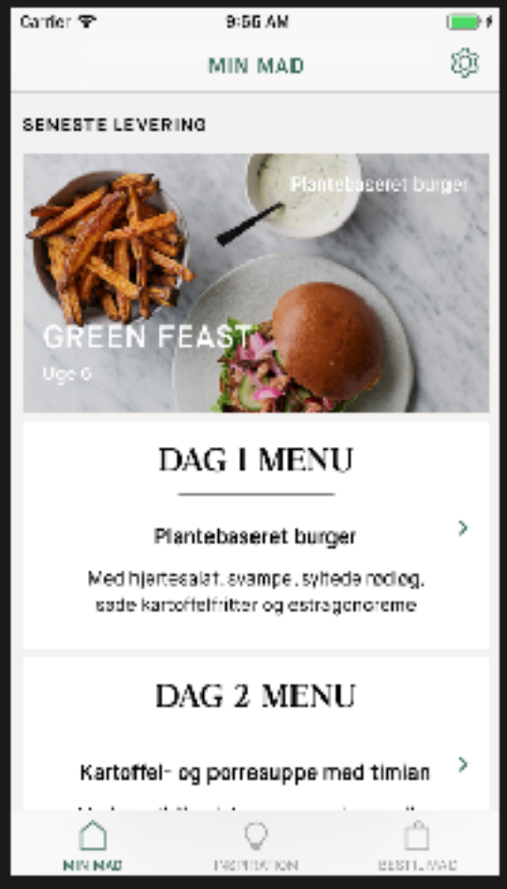
Feast

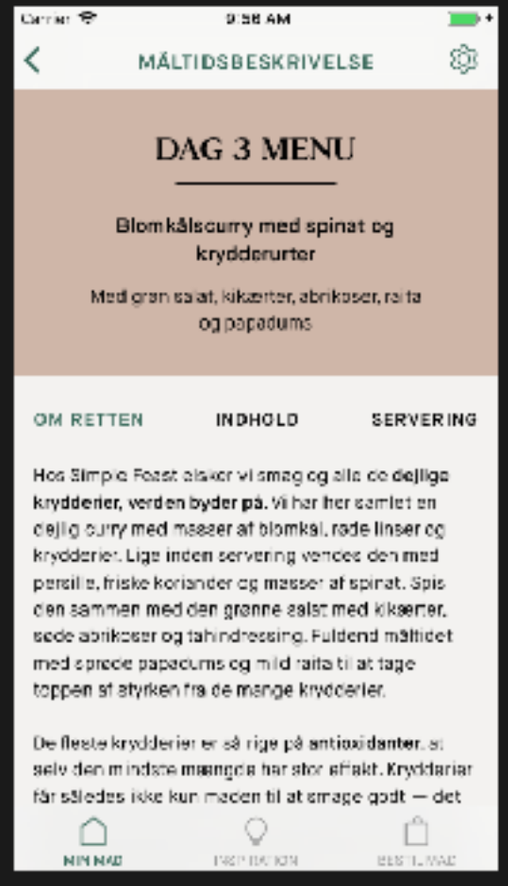
THE GOOD FOOD COMPANY

- Food-tech company
- Digital cookbook and meal kit service
- Vegetarian / Vegan meals
- Cooks in 10 mins
- 100% Organic
- Sustainable packaging


BESTIL MAD
GREEN FEAST
Ugentlige vegetariske måltider
VEGAN FEAST
Ugentlige veganske måltider
MIN MAD
INSPIRATION
BESTIL MAD


BESTIL MAD
VEGAN FEAST
Ugentlige veganske måltider
Antal personer:
4-5 personer
Kassen skal leveres:
Hver uge
DKK63 / portion
DKK945 (+ fragt)
BESTIL
Tre lækre, vegetarske og 100 % økologiske måltider om ugen, hvor det eneste du skal gøre er at færdiggøre og servere menuen. Alt er tilberedt for at give dig den lækreste, sundeste og mest bæredygtige aftensmad. Du kan afbestille når som helst.
DIN KOMMENDE LEVERING
MIN MAD
INSPIRATION
BESTIL MAD


MIN MAD
SENESTE LEVERING
Plantebaseret burger
GREEN FEAST
Uge 6
DAG 1 MENU
Plantebaseret burger
Med hjertesalat, svampe, syltede rødløg, søde kartoffelfritter og estragonremoulade
DAG 2 MENU
Kartoffel- og porresuppe med timian
MIN MAD
INSPIRATION
BESTIL MAD


MÅLTIDSBESKRIVELSE
DAG 3 MENU
Blomkålscurry med spinat og krydderurter
Med grøn salat, kikærter, abrikoser, raita og papadums
OM RETTEN
INDHOLD
SERVERING
Hos Simple Feast elsker vi smag og alle de dejlige krydderier, verden byder på. Vi har her samlet en dejlig curry med masser af blomkål, røde linser og krydderier. Lige inden servering vendes den med persille, friske koriander og masser af spinat. Spis den sammen med den grønne salat med kikærter, søde abrikoser og tahindressing. Fuldend måltidet med sprøde papadums og mild raita til at tage toppen af styrken fra de mange krydderier.
De fleste krydderier er så rige på antioxidanter, at selv den mindste mængde har stor effekt. Krydderier får således ikke kun maden til at smage godt — det
MIN MAD
INSPIRATION
BESTIL MAD

Requirements

- Some structure / organising principle
- Separation of concern
- Flexible
- Testable
- Keeping it simple

Some patterns

MVVM

- Some separation of concern
- Testable
- Where does navigation go?
- Massive View Model

VIPER

- Clear separation of concern
- Clearly defined structure, strict, not very flexible
- Complex, cognitive overload

Idea: A Story

- Defines a "story" or use case
- Knows how to navigate within that story
- Centralised knowledge of flow
- Isolate navigation from other entities

Back in 2015/16

Soroush Khanlou - Coordinators

http://khanlou.com/2015/10/coordinators-redux

Krzysztof Zabłocki - Flow Controllers

http://merowing.info/2016/01/improve-your-ios-architecture-with-flowcontrollers

MVVM

Model

- Raw data, no logic
- May update dynamically, i.e. core data, realm etc.
- Isolated, may know of other models, i.e. relationships)

View Model

- Presentation logic, i.e. when should stuff change?
- Transforms raw data into something a user can read
- Binds to model, updates based on changes
- Does not know about UIKit

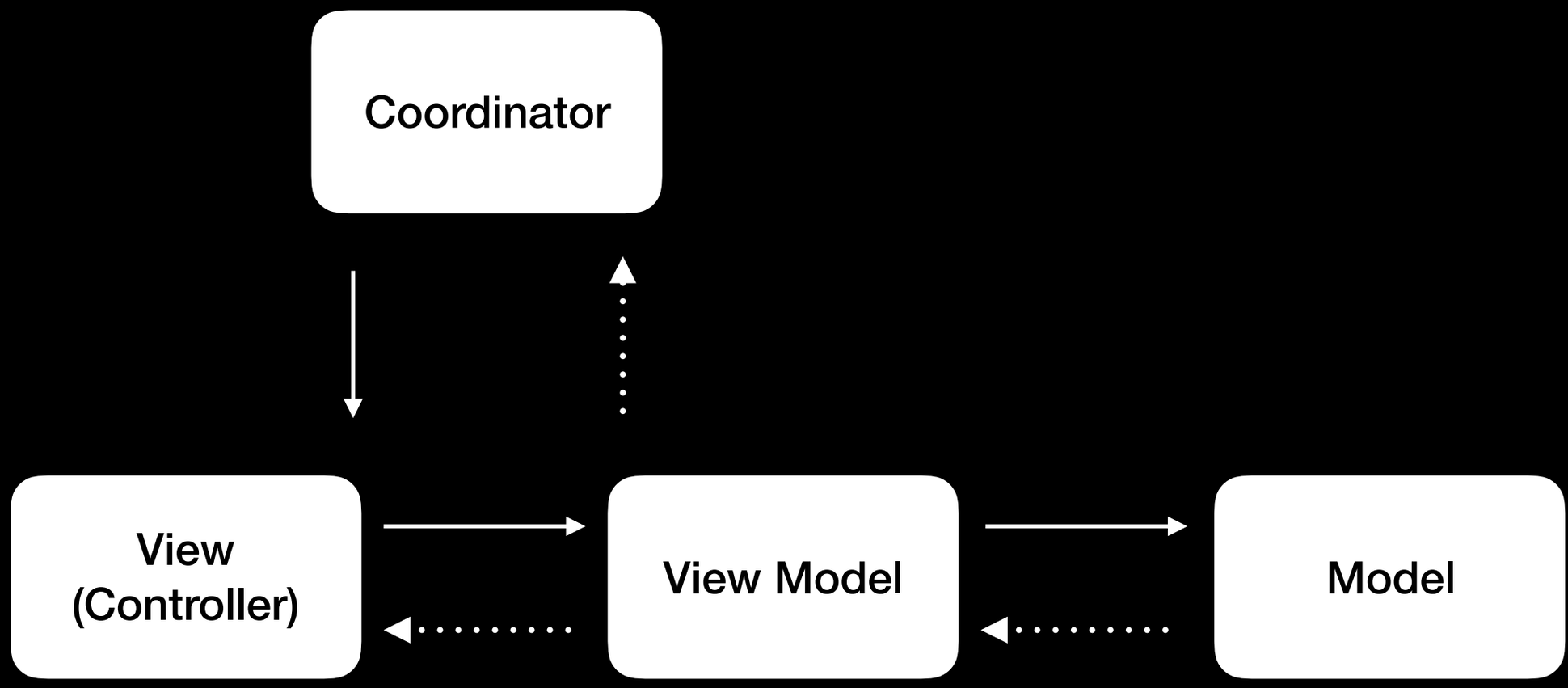
View (Controller)

- Layout logic, i.e. how do things look?, where are they placed?
- Binds to view model, updates based on view model changes
- Forwards user input to view model

The coordinator

- Navigation logic, defines a story or use case
- How are views stringed together within a "story"?
- Performs UIKit methods like push / present / dismiss
- Initialise and configure UIViewControllers
- Binds to view models, reacts to events, directs the flow
- Can present other coordinators

MVVMC



Example

An app with two use cases

- Browse beer use case
- Purchase beer use case

```swift
import UIKit

/// A protocol defining an interface for coordinators.
protocol Coordinating: class {

    /// The root view controller of the coordinator.
    var rootViewController: UINavigationController { get }

    /// The list of child coordinators of the coordinator.
    var childCoordinators: [Coordinating] { get set }

    /// Start the coordinator.
    func start()
}
```

```swift
extension Coordinating {

    /// Add the given coordinator to the list of child coordinators.
    func addChildCoordinator(childCoordinator: Coordinating) {
        childCoordinators.append(childCoordinator)
    }

    /// Remove the given coordinator from the list of child coordinators.
    func removeChildCoordinator(childCoordinator: Coordinating) {
        childCoordinators = childCoordinators.filter { $0 !== childCoordinator }
    }
}
```

```swift
import UIKit

/// A coordinator defining the "story" of browsing beers.
class BrowseBeerCoordinator: Coordinating {

    var rootViewController: UINavigationController
    var childCoordinators: [Coordinating]

    init() {
        self.rootViewController = UINavigationController()
        self.childCoordinators = []
    }

    func start() {
        // Show the beer list as the first view of the coordinator.
        showBeerList()
    }
}

private func showBeerList() {

    // Setup and configure the view controller.
    let viewController = BeerListViewController.storyboardInstance()
    configure(viewController: viewController)

    // Push the view controller on the navigation stack of the coordinator.
    rootViewController.pushViewController(viewController, animated: true)
}
```

```swift
protocol BeerListViewModelProtocol {

    /// The title of the view.
    var title: String { get }

    /// Select the beer at the given index path.
    func selectBeer(at indexPath: IndexPath)
}

protocol BeerListViewModelEvents {

    /// Invoked when the given beer was selected.
    var beerSelected: (Beer) -> Void { get set }
}
```

```swift
private func configure(viewController: BeerListViewController) {

    // Configure the view controller with its view model.
    let viewModel = BeerListViewModel()
    viewController.viewModel = viewModel

    // Inject the next step in the flow into the view model as a closure.
    viewModel.beerSelected = { [unowned self] beer in
        self.showBeerDetails(for: beer)
    }
}
```

```swift
private func presentBuyBeerCoordinator(beer: Beer, count: Int) {

    // Setup coordinator and inject flow control as a closure.
    let coordinator = BuyBeerCoordinator(beer: beer, count: count)
    coordinator.purchaseCompleted = { [unowned self] buyBeerCoordinator in

        // On purchase complete, remove the coordinator and dismiss its rootViewController.
        self.removeChildCoordinator(childCoordinator: buyBeerCoordinator)
        buyBeerCoordinator.rootViewController.dismiss(animated: true)
    }

    // Add the coordinator as a child and present its root view controller
    addChildCoordinator(childCoordinator: coordinator)
    coordinator.start()

    rootViewController.present(coordinator.rootViewController, animated: true)
}
```

MVVMC

- Isolates and centralises navigation
- ViewControllers and ViewModels are isolated, reuse
- Focus on one thing at a time, i.e. layout, presentation, navigation
- Good balance between structure and flexibility
- Healthy separation of concern
- While not overdoing architecture

Questions?