Assignment: Asynchronous Data Loading in Vives Plus App

Course: iOS Development
Program: Bachelor Applied Computer Science
Topic: Async/Await and Data Loading with SwiftUI
Instructors: Milan Dima, Dirk Hostens

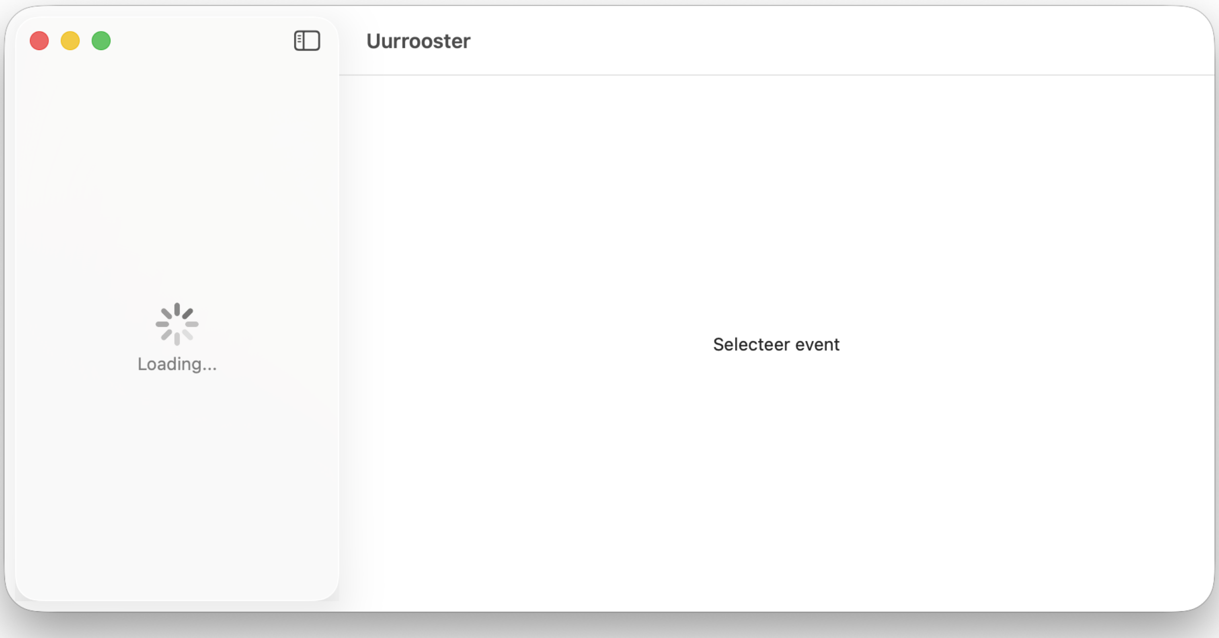
📋 Introduction

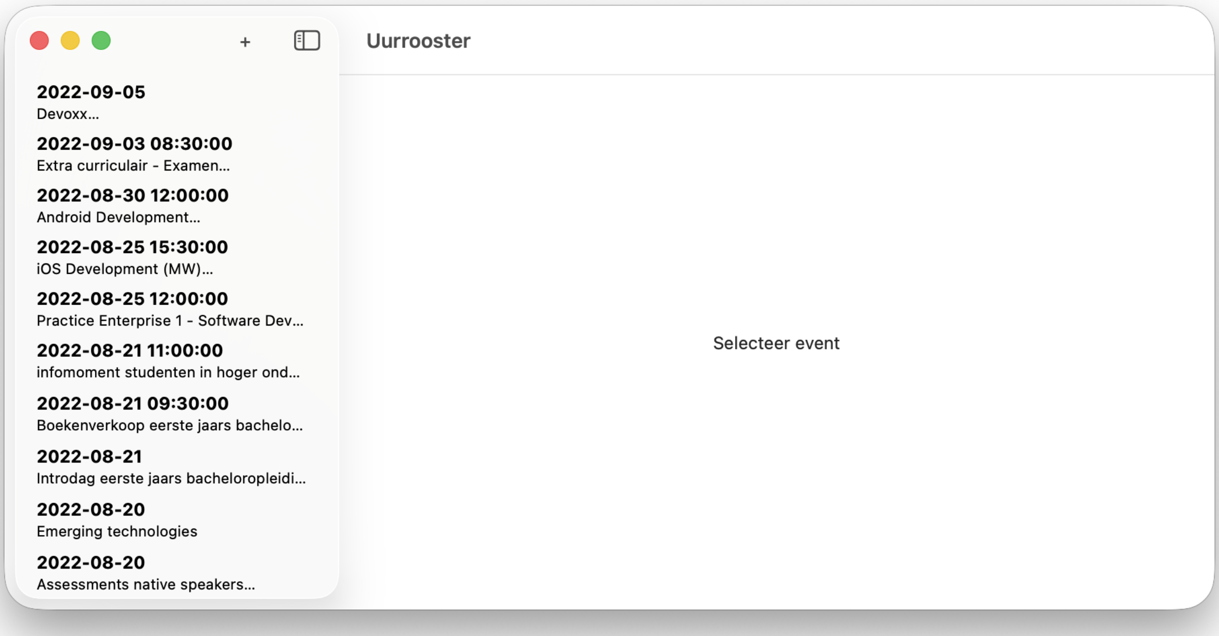
In this assignment you will learn how to work with **asynchronous data loading** in SwiftUI. You will apply the `loadData()` function from the Schedule app to your own **Vives Plus** application. This technique is essential when loading data from external sources such as JSON files, REST APIs, or databases.

Functionality of the Schedule App

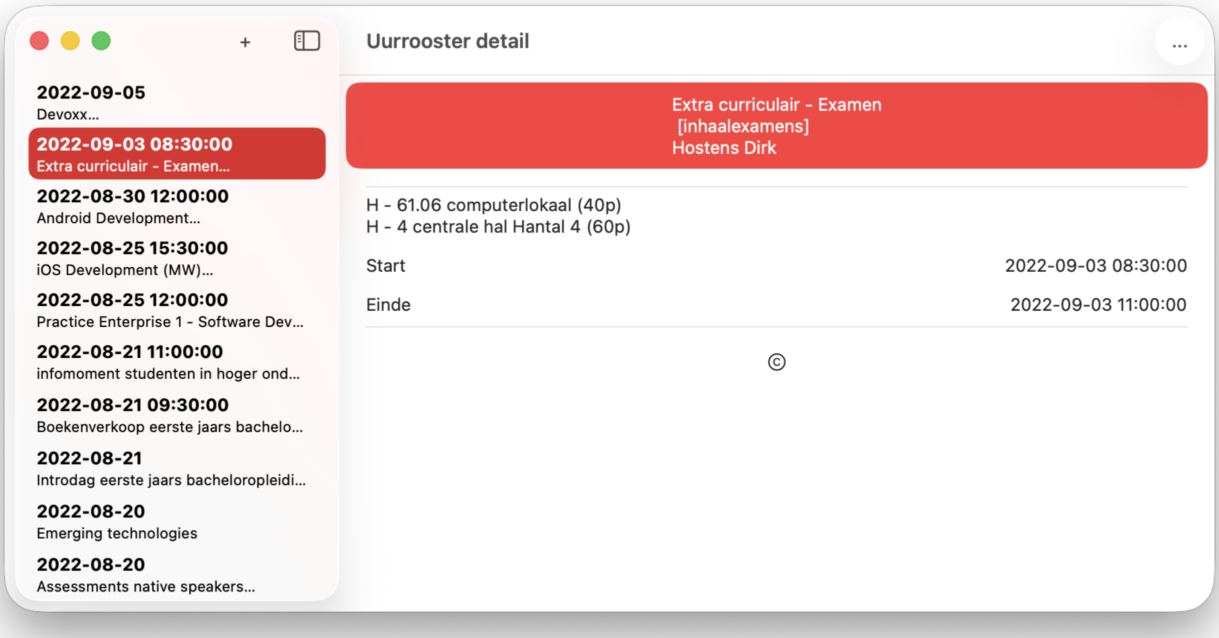
The Schedule app offers the following functionalities:

- **List of events**: Events are loaded asynchronously and displayed in a list
- **Add Event**: Via the action bar button you can add new events
- **Edit Event**: Via the action bar button you can edit existing events
- Events are sorted by date (newest first)





2022-09-05
Devoxx...
2022-09-03 08:30:00
Extra curriculair - Examen...
2022-08-30 12:00:00
Android Development...
2022-08-25 15:30:00
iOS Development (MW)...
2022-08-25 12:00:00
Practice Enterprise 1 - Software Dev...
2022-08-21 11:00:00
infomoment studenten in hoger ond...
2022-08-21 09:30:00
Boekenverkoop eerste jaars bachelo...
2022-08-21
Introdag eerste jaars bacheloropleidi...
2022-08-20
Emerging technologies
2022-08-20
Assessments native speakers...



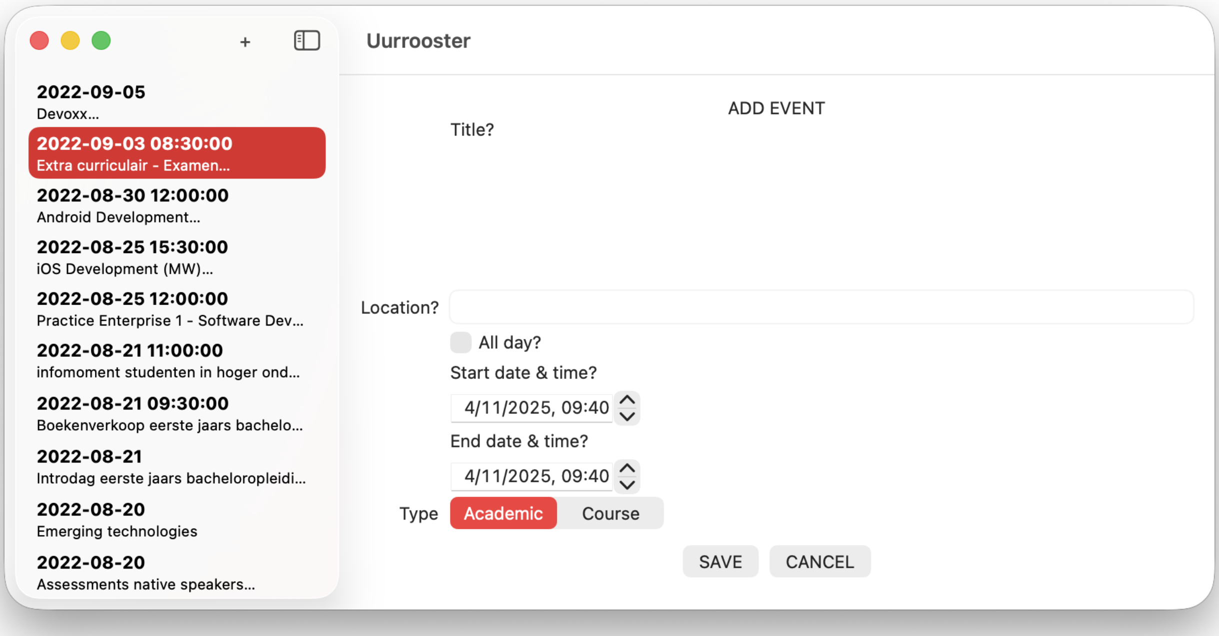
Uurrooster detail

2022-09-05	Devoxx...
2022-09-03 08:30:00	Extra curriculair - Examen...
2022-08-30 12:00:00	Android Development...
2022-08-25 15:30:00	iOS Development (MW)...
2022-08-25 12:00:00	Practice Enterprise 1 - Software Dev...
2022-08-21 11:00:00	infomoment studenten in hoger ond...
2022-08-21 09:30:00	Boekenverkoop eerste jaars bachelo...
2022-08-21	Introdag eerste jaars bacheloropleidi...
2022-08-20	Emerging technologies
2022-08-20	Assessments native speakers...

Extra curriculair - Examen
[inhaalexamens]
Hostens Dirk

H - 61.06 computerlokaal (40p)
H - 4 centrale hal Hantal 4 (60p)

Start	2022-09-03 08:30:00
Einde	2022-09-03 11:00:00



Uurrooster

2022-09-05	Devoxx...
2022-09-03 08:30:00	Extra curriculair - Examen...
2022-08-30 12:00:00	Android Development...
2022-08-25 15:30:00	iOS Development (MW)...
2022-08-25 12:00:00	Practice Enterprise 1 - Software Dev...
2022-08-21 11:00:00	infomoment studenten in hoger ond...
2022-08-21 09:30:00	Boekenverkoop eerste jaars bachelo...
2022-08-21	Introdag eerste jaars bacheloropleidi...
2022-08-20	Emerging technologies
2022-08-20	Assessments native speakers...

ADD EVENT

Title?

Location?

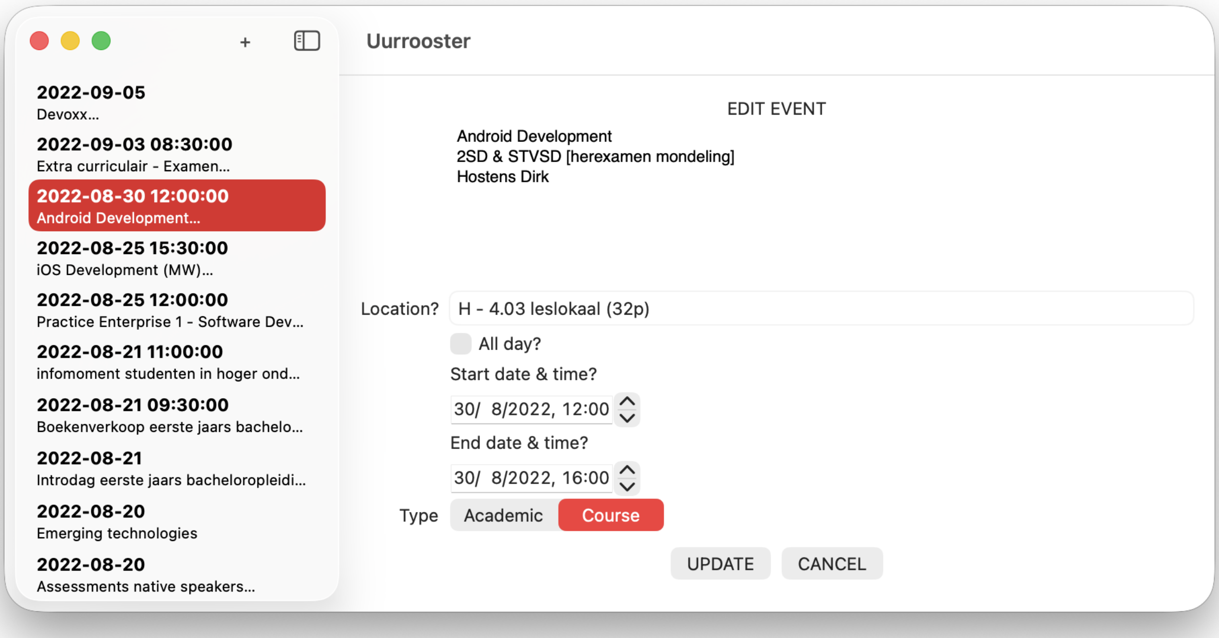
All day?

Start date & time? 4/11/2025, 09:40

End date & time? 4/11/2025, 09:40

Type [✓ Academic] Course

SAVE CANCEL



2022-09-05
Devoxx...
2022-09-03 08:30:00
Extra curriculair - Examen...
2022-08-30 12:00:00
Android Development...
2022-08-25 15:30:00
iOS Development (MW)...
2022-08-25 12:00:00
Practice Enterprise 1 - Software Dev...
2022-08-21 11:00:00
infomoment studenten in hoger ond...
2022-08-21 09:30:00
Boekenverkoop eerste jaars bachelo...
2022-08-21
Introdag eerste jaars bacheloropleidi...
2022-08-20
Emerging technologies
2022-08-20
Assessments native speakers...

Android Development
2SD & STVSD [herexamen mondeling]
Hostens Dirk

Location? H - 4.03 leslokaal (32p)
All day?
Start date & time?
30/ 8/2022, 12:00
End date & time?
30/ 8/2022, 16:00
Type Academic [✓ Course]

🎯 Learning Objectives

After this assignment you will be able to:

- ✅ Explain how async/await works in Swift
- ✅ Implement asynchronous functions in a DataStore class
- ✅ Add a loading state to your SwiftUI views
- ✅ Use the `.task` modifier for async operations
- ✅ Load JSON data and map it to model objects

📚 Background: The ScheduleDataStore Class

Overview of the ScheduleDataStore

The `ScheduleDataStore` class is an **Observable** data model that acts as the single source of truth for the application. Here is an overview of all functions:

Function	Purpose	Parameters	Return
`init()`	Initializes an empty schedule array	-	-
`sort()`	Sorts events by startDateTime (newest first)	-	-
`addEvent(event:)`	Adds new event with UUID	EventModel	-
`updateEvent(event:)`	Updates existing event based on id	EventModel	-
`deleteEvent(id:)`	Removes event with given id	String	-
`getEvent(id:)`	Retrieves specific event	String	EventModel

Function	Purpose	Parameters	Return
`loadData()`	**Loads data asynchronously**	-	async

The loadData() Function in Detail

```
func loadData() async {
    //simulate async call
    do {
        print("⏳ Simulating 2-second load delay...")
        try await Task.sleep(for: .seconds(2)) // Simulate long load
        let data: [EventModelJson] = try load("schedule.json")
        // Mapping to EventModel goes here
        sort()
        print("✅ Data loaded successfully.")

    } catch {
        print("❌ Failed to load schedule:", error)
        schedule = [EventModel]()
    }
}
```

What Does This Function Do?

1. `async` **keyword**: Marks the function as asynchronous - it can wait without blocking the UI
2. `Task.sleep(for: .seconds(2))`: Simulates a delay (like during a network call)
3. `try await`: Waits for the async operation and catches possible errors
4. `load("schedule.json")`: Loads JSON data from the bundle
5. `.map()`: Converts JSON objects to EventModel objects
6. `sort()`: Sorts the loaded events
7. **Error handling**: On error, an empty array is set

Why Async?

- 🚫 **Without async**: The app would freeze during loading
- ✅ **With async**: The UI remains responsive, user sees loading indicator

---

🔧 Usage in ScheduleList View

Step-by-Step Explanation

1️⃣ Loading State

```
@State var loading = true
```

- Tracks whether data is still loading

- Starts at `true` because data hasn't been loaded yet
- Triggers UI update when value changes

2️⃣ Conditional Rendering

```
if loading {
    ProgressView("Loading...")
} else {
    List(scheduleDataStore.schedule, ...) { ... }
}
```

- **During loading**: Show ProgressView (spinner)
- **After loading**: Show the list of events

3️⃣ Task Modifier

```
.task {
    await scheduleDataStore.loadData()
    loading = false
}
```

- `.task`: SwiftUI modifier that executes async code
- Automatically called when view appears
- `await`: Waits until loadData() is finished
- `loading = false`: Updates state to show list
- With the `.task` modifier you can directly call the async function `loadData()` as demonstrated above

4️⃣ DataStore as Environment Object

The `DataStore` class must be added as an `@Environment` object to your view:

```
@Environment(ScheduleDataStore.self) private var scheduleDataStore
```

This ensures your DataStore is available throughout your view hierarchy.

5️⃣ Toolbar with NavigationLink

To add navigation buttons in the toolbar (e.g., for adding or editing events), use the `.toolbar` modifier:

```
.toolbar {
    NavigationLink(destination: AddEventView()) {
        Image(systemName: "plus")
        }
    }
```

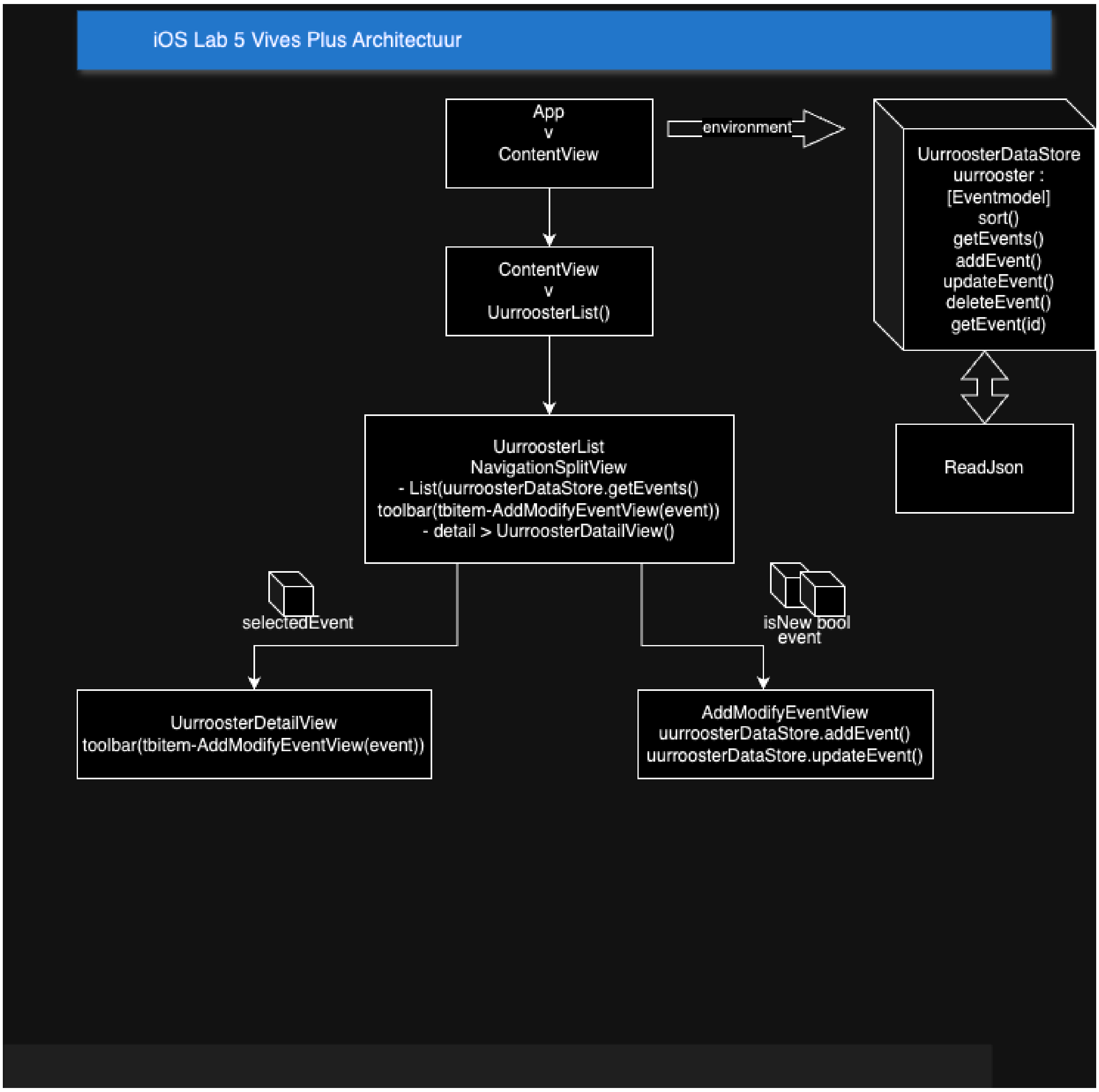
- `.toolbar`: Adds a toolbar to your navigation view
- `NavigationLink`: Creates a button that navigates to another view when tapped
- `destination:`: The view to navigate to when the button is pressed
- `Image(systemName:)`: SF Symbol icon for the button (e.g., "plus" for adding)
- The NavigationLink will automatically be placed in the trailing position of the navigation bar

---

📖 Useful Resources

- Swift Concurrency Documentation
- SwiftUI Task Modifier
- Observable Macro

Architecture


iOS Lab 5 Vives Plus Architectuur
App
v
ContentView
environment
UurroosterDataStore
uurrooster :
[Eventmodel]
sort()
getEvents()
addEvent()
updateEvent()
deleteEvent()
getEvent(id)
ContentView
v
UurroosterList()
ReadJson
UurroosterList
NavigationSplitView
- List(uurroosterDataStore.getEvents()
toolbar(tbitem-AddModifyEventView(event))
- detail > UurroosterDatailView()
selectedEvent
isNew bool
event
UurroosterDetailView
toolbar(tbitem-AddModifyEventView(event))
AddModifyEventView
uurroosterDataStore.addEvent()
uurroosterDataStore.updateEvent()

Good luck! 🎉