# Design-driven DEVELOPMENT

by Craig Clayton

/designtoswiftui

thedevme

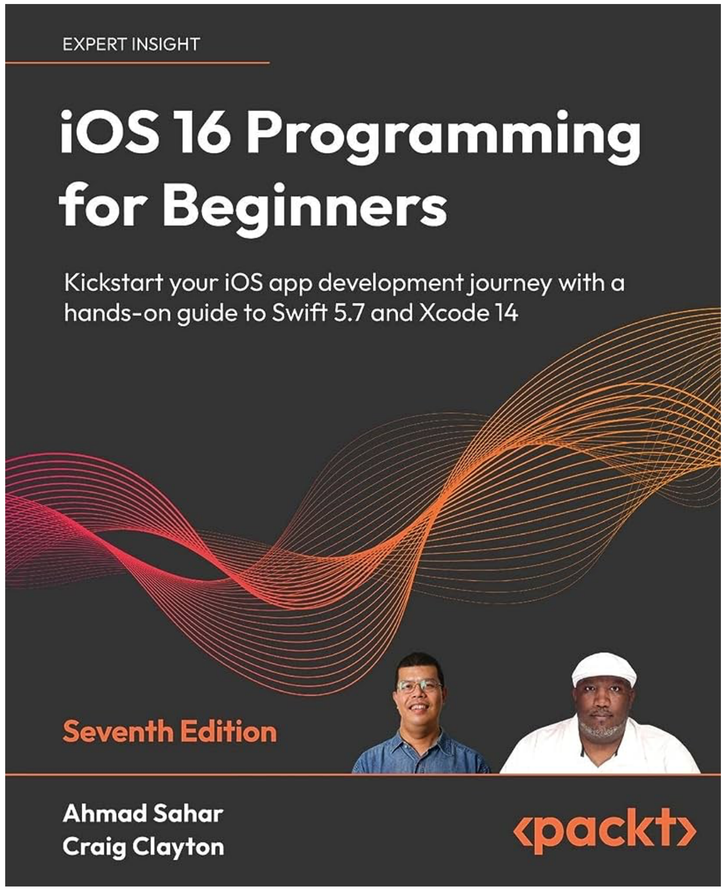
mastodon.cloud/@thedevme

DS {to}

# A LIL ABOUT
# ME

I AM

# SELF-TAUGHT

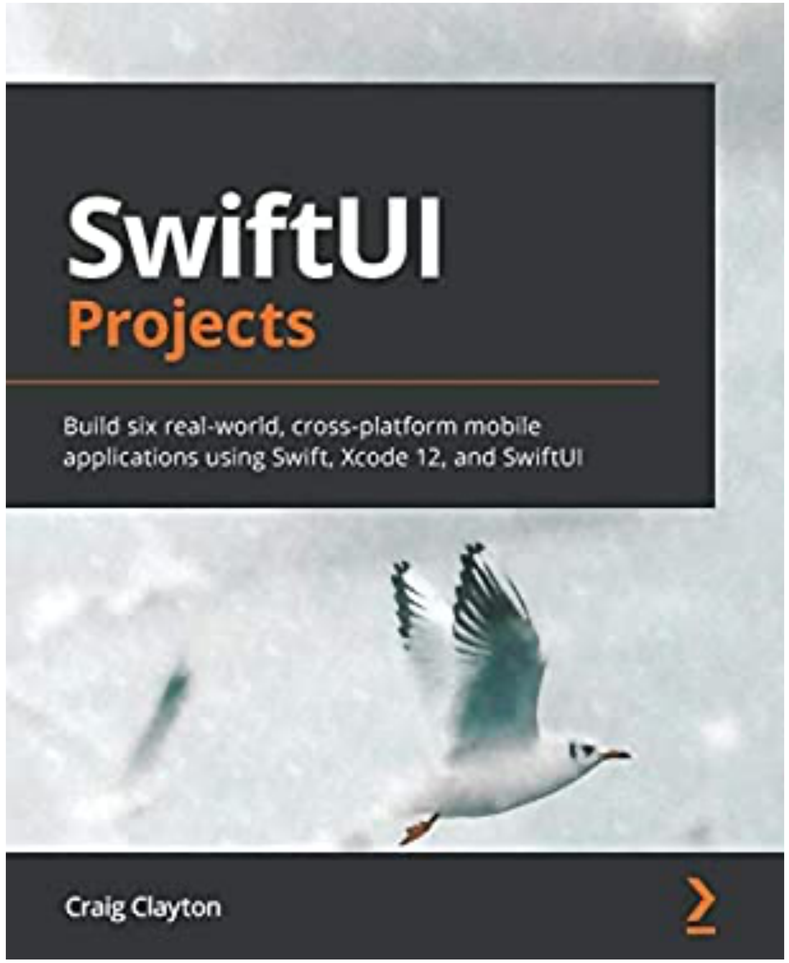



# LAST VERSION...

thedevme

mastodon.cloud/
@thedevme

/designtoswiftui




thedevme

mastodon.cloud/
@thedevme

/designtoswiftui

DS {to}

YouTube

/designtoswiftui

# WHAT WE WILL COVER

- The backstory
- Inside vs Outside
- User stories
- Practical example
- Wrapping up

DS {to}

# THE BACKSTORY

1

# BUT

**Mikaela Caron**
**@mikaela__caron**

PORZINGIS
6

PORZINGIS
6

#DiffrentStrokes

PayPal

- Kevin Durant

He can shoot, he can make the right plays, he can defend, he’s a 7-footer that can shoot all the way out to the 3-point line. That’s rare. And block shots—that’s like a unicorn in this league."

- Kevin Durant

# DESIGN & DEVELOP

Fl

# 2008

# 2010

# 2024

**Craig Clayton**
@thedevme · Jan 7 ·

I am determined to use Unit testing in iOS in 2024. Please give me your recommendations for YT videos, courses, and books. And any other things you would recommend.

Thank you! 🙏

7 5

Replied to Craig Clayton

**Chris Wu** 🍌

@MuseumShuffle@mastodon.social · Jan 7 ·

@thedevme Watching this thread very closely. 😬

2 1

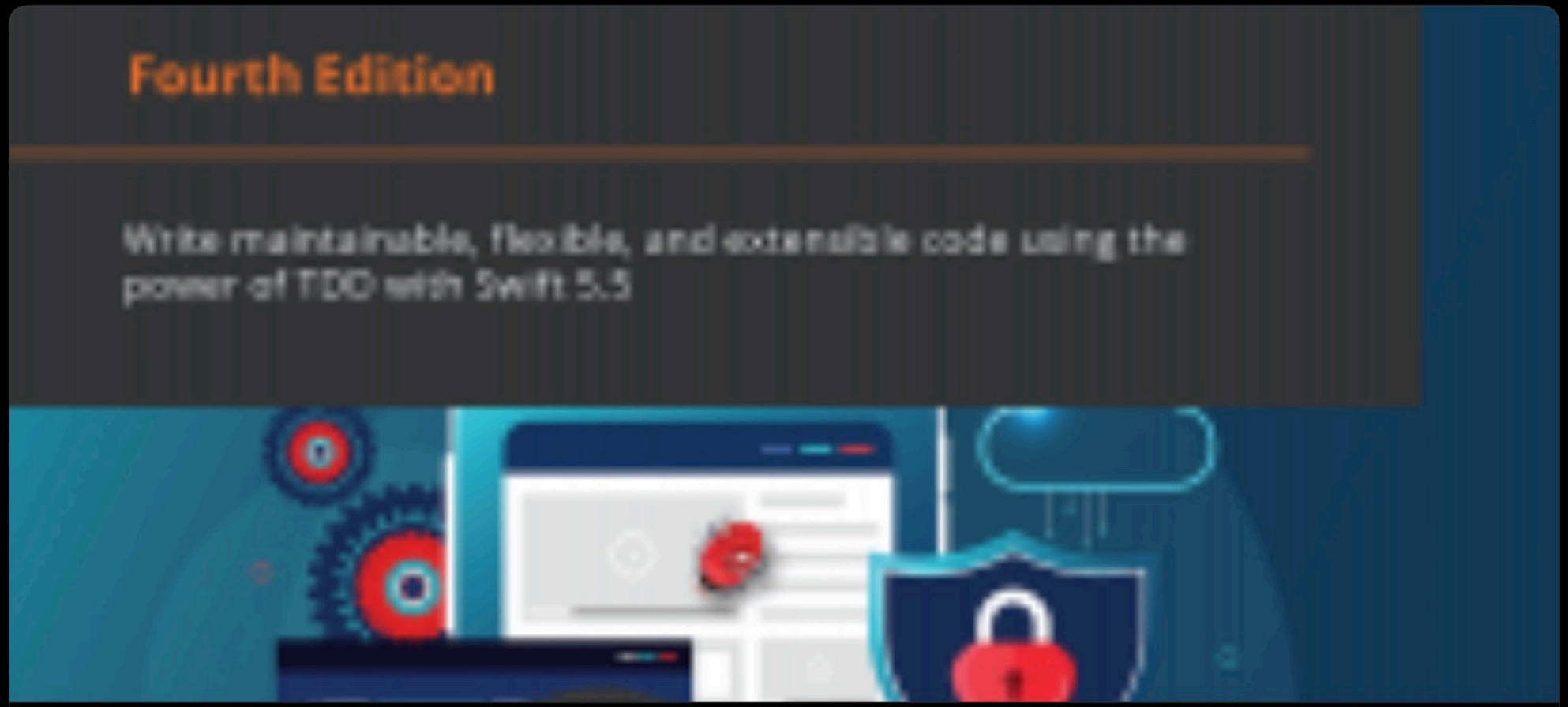
Favorites

Jan 7, 2024 at 11:25 AM

Replied to Craig Clayton

**dasdom**
@dasdom@chaos.social · Jan 7 ·

@thedevme If you get a discount at Packt, they still sell my book about TDD.
packtpub.com/product/test-driv...




Test-Driven iOS Development with Swift - Fourth Edition | Packt
www.packtpub.com

1 1

Replied to Craig Clayton

**Ellen Shapiro**

@designatednerd@mastodon.social · Edited Jan 8 ·

@thedevme @qcoding @dasdom for me I think behavioral testing gets more to the meat of what we're trying to do: Make things work for the user.

If you want to adopt a test first mentality, personally I find it a lot easier to start there than at unit tests (which I find tend to come naturally from breaking down the behaviors to their component parts).

1

Replied to Ellen Shapiro

**Jon Reid**
@qcoding@iosdev.space · Jan 8 ·

@designatednerd @thedevme @dasdom I'd love to learn what _you_ mean by BDD. It sounds like you want some kind of customer-facing acceptance tests. Do you like to express these as UI tests, or table-driven tests of actions in order and expected results, or just readable test code, or...?

1

Replied to Craig Clayton

**Jon Reid**
@qcoding@iosdev.space · Jan 9 ·

@thedevme @designatednerd One thing that keeps puzzling me: I read what you both describe, Craig and Ellen. Both sound like they could be TDD.

So where's the disconnect? It could be that if you've watched me, I usually favor inside-out TDD (aka Chicago school). But there's another style, outside-in TDD (aka London school). The really good TDDers mix-and-match.

So outside-in TDD sounds like it would fit you better.

3

2:15 AM Wed Jan 17 100%

SEARCH FOR A PLAYER

SORT PLAYERS BY

PTS REB AST

FRONTCOURT EASTERN PHI

15 REAVES, AUSTIN
WEST • LAL • GUARD
15.2 PTS 5.1 AST 4.3 REB

2 GILGEOUS-ALEXANDER, SHAI
WEST • OKC • GUARD
31.5 PTS 6.4 AST 5.9 REB

23 JAMES, LEBRON
WEST • LAL • FRONTCOURT
24.7 PTS 7.6 AST 7.2 REB

3 DAVIS, ANTHONY
WEST • LAL • FRONTCOURT
25.1 PTS 3.5 AST 12.1 REB

1 WEMBANYAMA, VICTOR
WEST • SAS • FRONTCOURT
19.4 PTS 3.0 AST 10.2 REB

0 11 21 34 0

I The I'm

DS {to}

# INSIDE VS. OUTSIDE

2

# INSIDE OUT

## CHICAGO STYLE

# OUTSIDE IN

## LONDON STYLE

# DS {to}

# PRACTICAL EXAMPLE

3

# AUDIENCE

## PARTICIPATION

You reposted

**mastodon.cloud/@t...** · 4/23/24

When starting a new app development project, which approach do you prefer? Do you begin by building out data models and core functionalities, or do you start with prototyping the user interface and design?
#AppDevelopment #Coding

| | |
|---|---|
| Data Models & Core | 49% |
| **UI & Design** | **51%** |

72 votes · Final results

6 4 5 3.9K

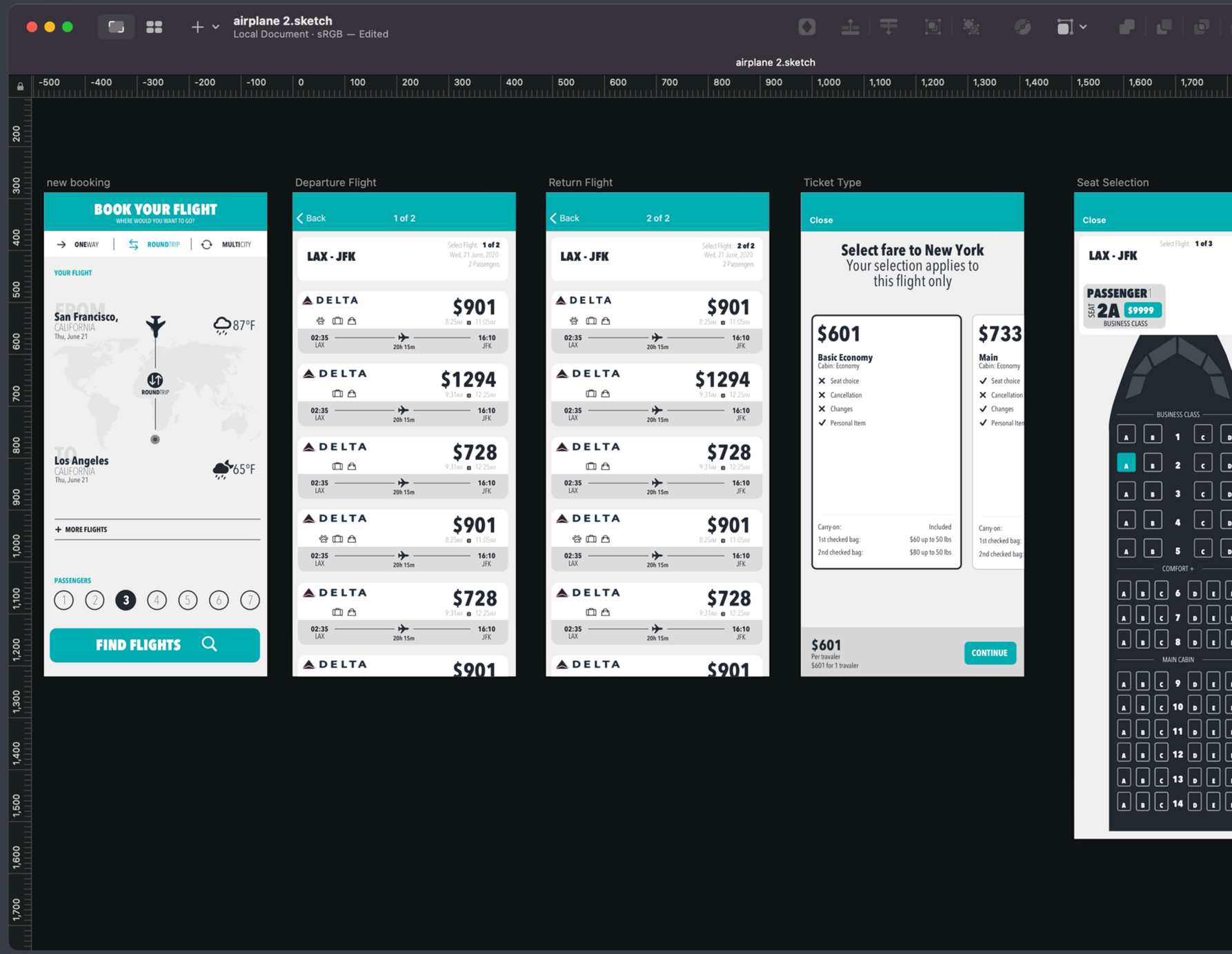
# AIRLINE TICKETING APP

EAST HAMPTON

STEP ONE

# DESIGN THE UI

# APP DESIGN

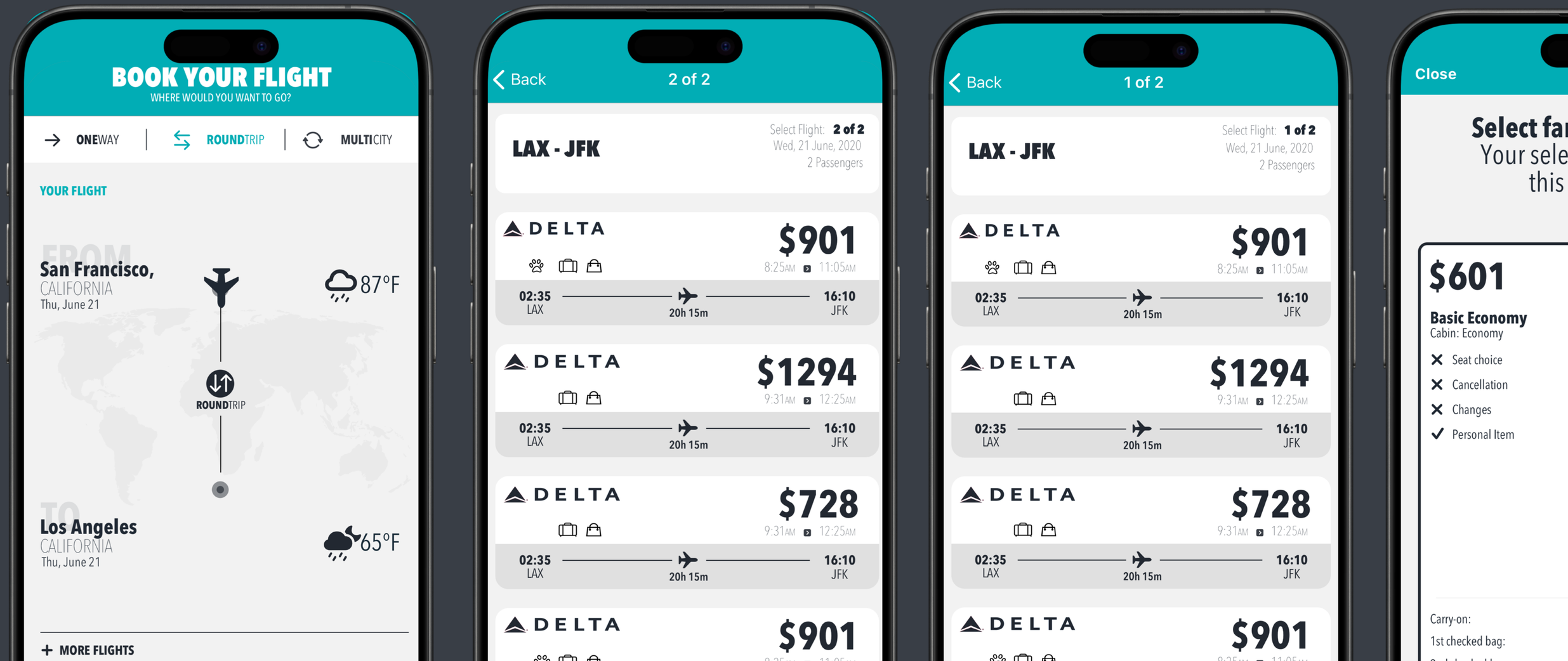

airplane 2.sketch
Local Document · sRGB — Edited
airplane 2.sketch
new booking
BOOK YOUR FLIGHT
WHERE WOULD YOU WANT TO GO?
ONEWAY
ROUNDTRIP
MULTICITY
YOUR FLIGHT
San Francisco,
CALIFORNIA
Thu, June 21
87°F
ROUNDTRIP
Los Angeles
CALIFORNIA
Thu, June 21
65°F
+ MORE FLIGHTS
PASSENGERS
1 2 3 4 5 6 7
FIND FLIGHTS
Departure Flight
Back
1 of 2
LAX - JFK
Select Flight: 1 of 2
Wed, 21 June, 2020
2 Passengers
DELTA $901
DELTA $1294
DELTA $728
DELTA $901
DELTA $728
DELTA $901
02:35 LAX
20h 15m
16:10 JFK
Return Flight
Back
2 of 2
LAX - JFK
Select Flight: 2 of 2
Wed, 21 June, 2020
2 Passengers
DELTA $901
DELTA $1294
DELTA $728
DELTA $901
DELTA $728
DELTA $901
02:35 LAX
20h 15m
16:10 JFK
Ticket Type
Close
Select fare to New York
Your selection applies to this flight only
$601
Basic Economy
Cabin: Economy
Seat choice
Cancellation
Changes
Personal Item
Carry-on: Included
1st checked bag: $60 up to 50 lbs
2nd checked bag: $80 up to 50 lbs
$733
Main
Cabin: Economy
Seat choice
Cancellation
Changes
Personal Item
Carry-on:
1st checked bag:
2nd checked bag:
$601
Per traveler
$601 for 1 traveler
CONTINUE
Seat Selection
Close
Select Flight: 1 of 3
LAX - JFK
PASSENGER 1
SEAT 2A $9999
BUSINESS CLASS
BUSINESS CLASS
COMFORT +
MAIN CABIN

STEP TWO

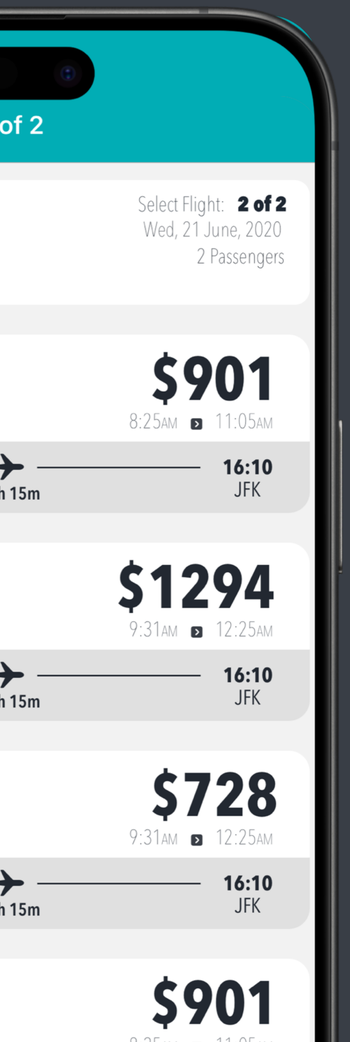
# PROTOTYPE IN

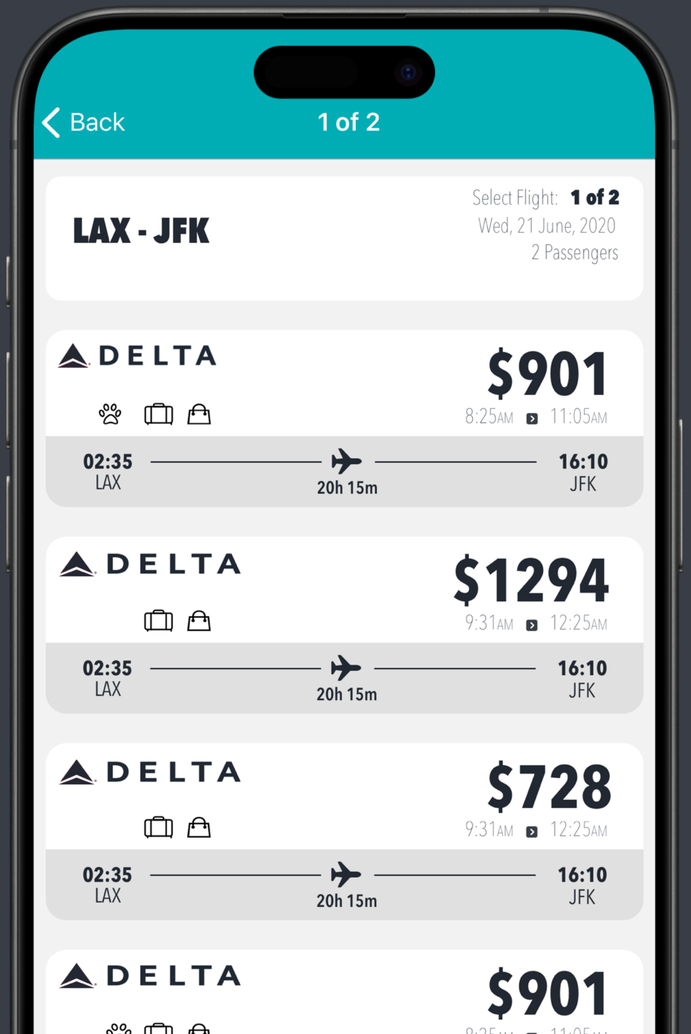
SWIFTUI

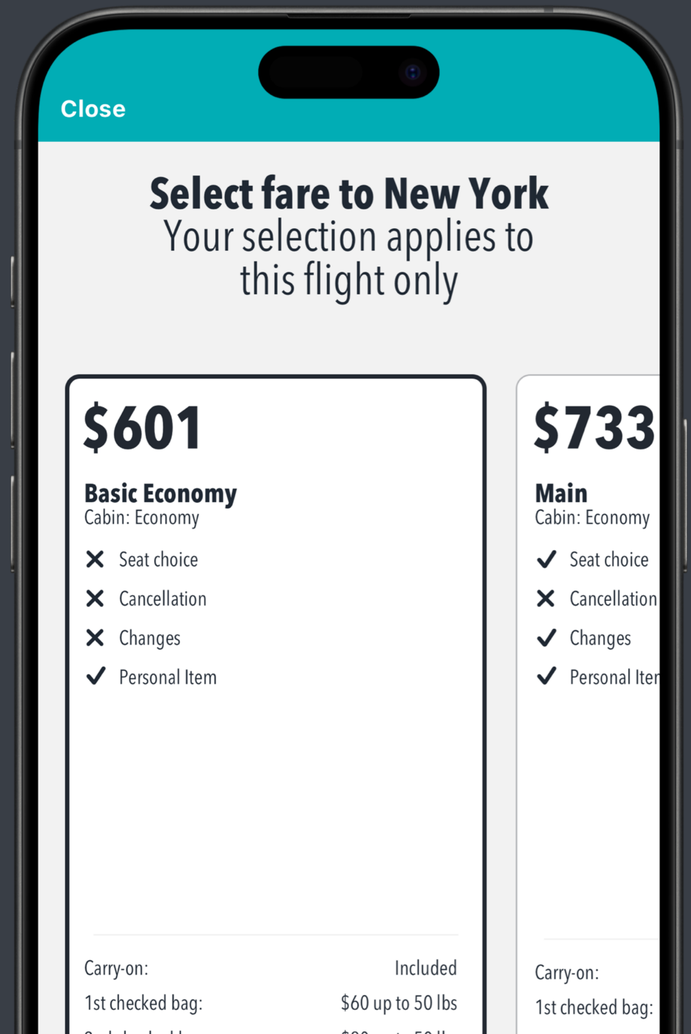
# AIRLINE TICKETING APP

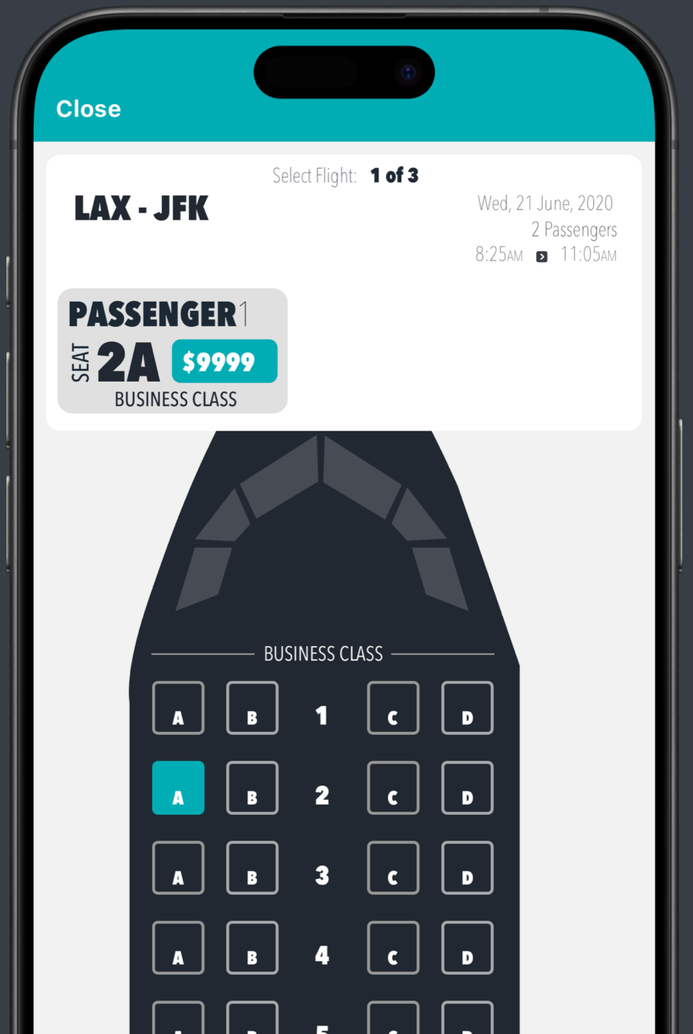


# AIRLINE TICKETING APP



STEP THREE

# DEFINE MOCKS

& STUBS

# WHAT ARE

## MOCKS?

# INSIDE OUT

# PURPOSE

## INSIDE OUT

# COMPONENT FOCUS

## INSIDE OUT

# DIRECTION

## INSIDE OUT

# OUTSIDE IN

# PURPOSE

## OUTSIDE IN

# USER FOCUS

## OUTSIDE IN

# DIRECTION

## OUTSIDE IN

# SUMMARY

# MOCK

```
struct MockSeat {
    let id: String
    let classType: SeatClass
    var isAvailable: Bool
    var isSelected: Bool

    enum SeatClass: String {
        case first, business, economy
    }
}
```

# WHAT ARE

## STUBS?

# INSIDE OUT

# STUBS

## INSIDE OUT

# FOCUS

## INSIDE OUT

# OUTSIDE IN

# STUBS

## OUTSIDE IN

# FOCUS

## OUTSIDE IN

# HOW THEY'RE

## DIFFERENT

# STUB

```
class SeatDataManagerStub {
    private(set) var seats: [MockSeat]

    init(seats: [MockSeat]) {
        self.seats = seats
    }
}
```

STEP FOUR

# WRITE UNIT TESTS

USING MOCKS & STUBS

# DS {to} USER STORY

4

GIVEN-WHEN-THEN

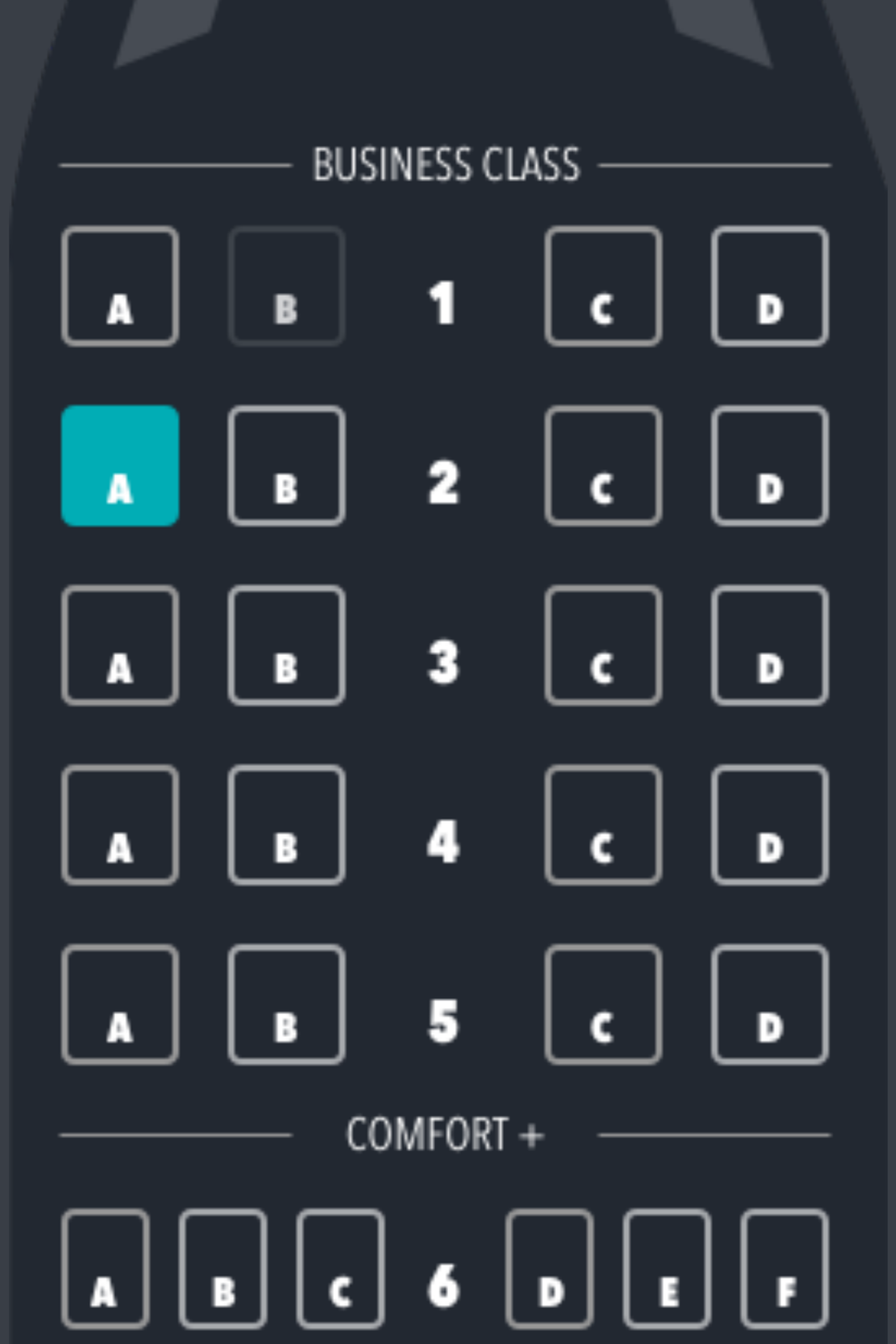
# BEHAVIOR-DRIVEN DEVELOPMENT

# GIVEN

# WHEN

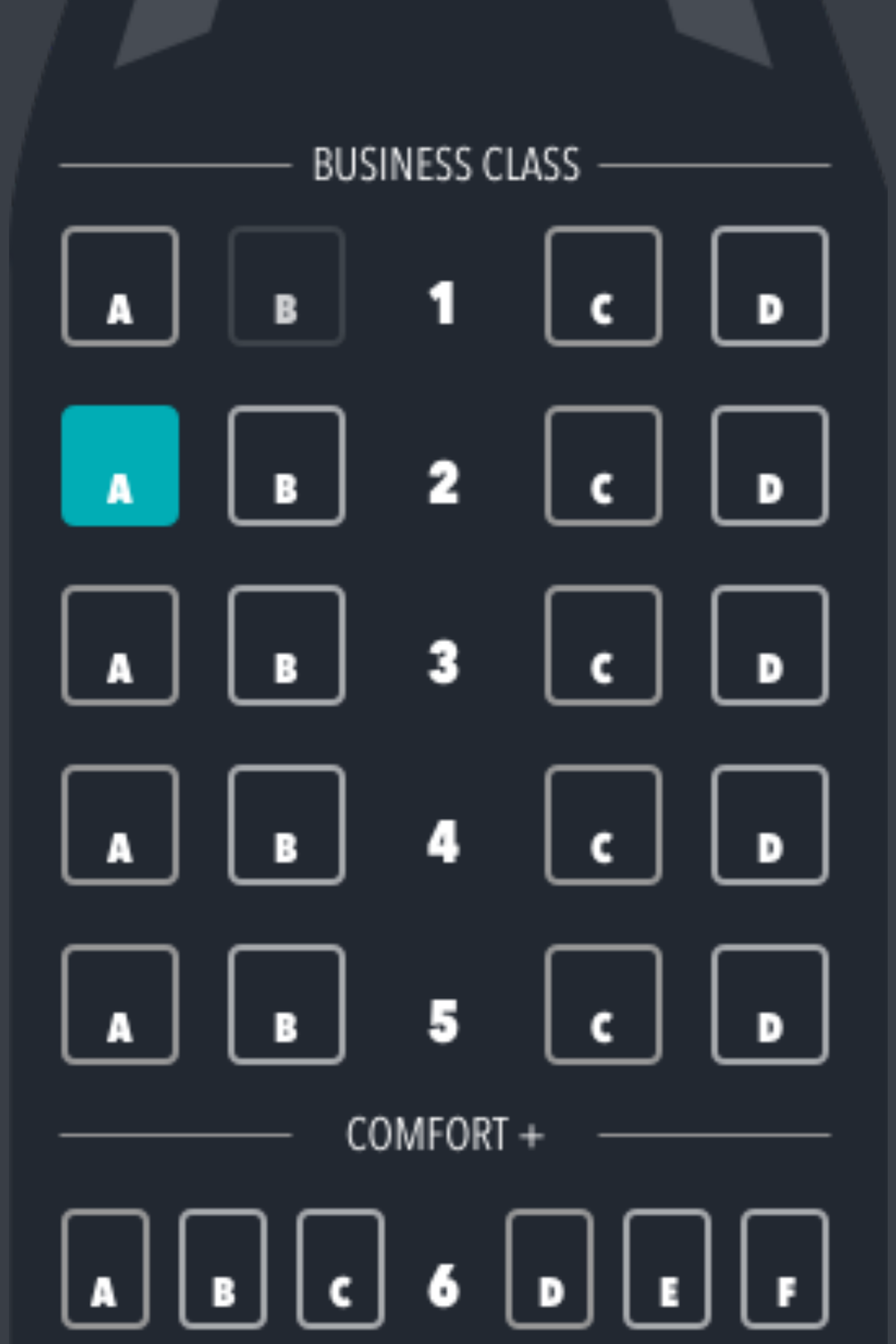
# THEN

# SCENARIO 1:
# Selecting an Available Seat



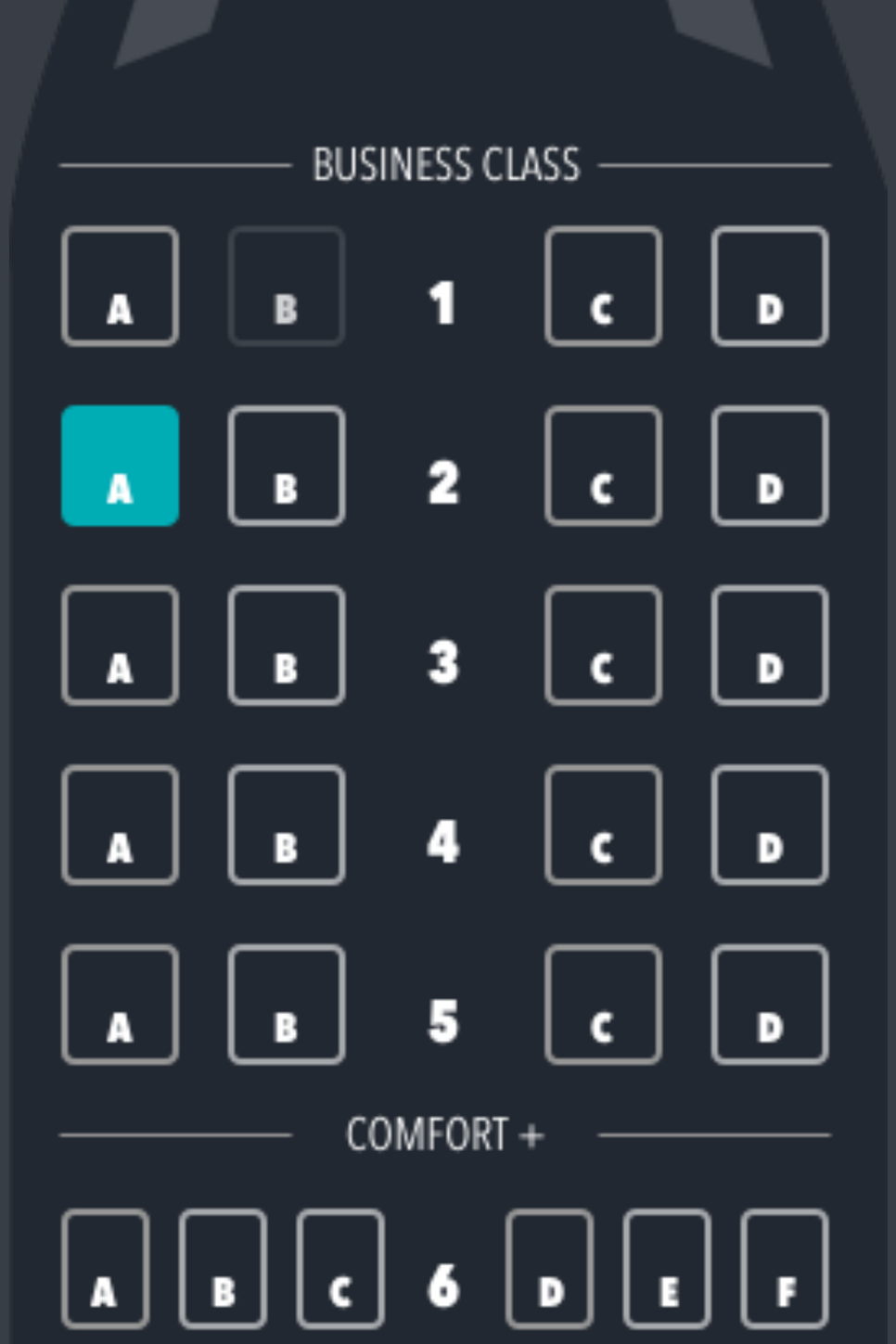
# SCENARIO 1:
# Selecting an Available Seat

- **Given:** The user is viewing an airplane seating chart.



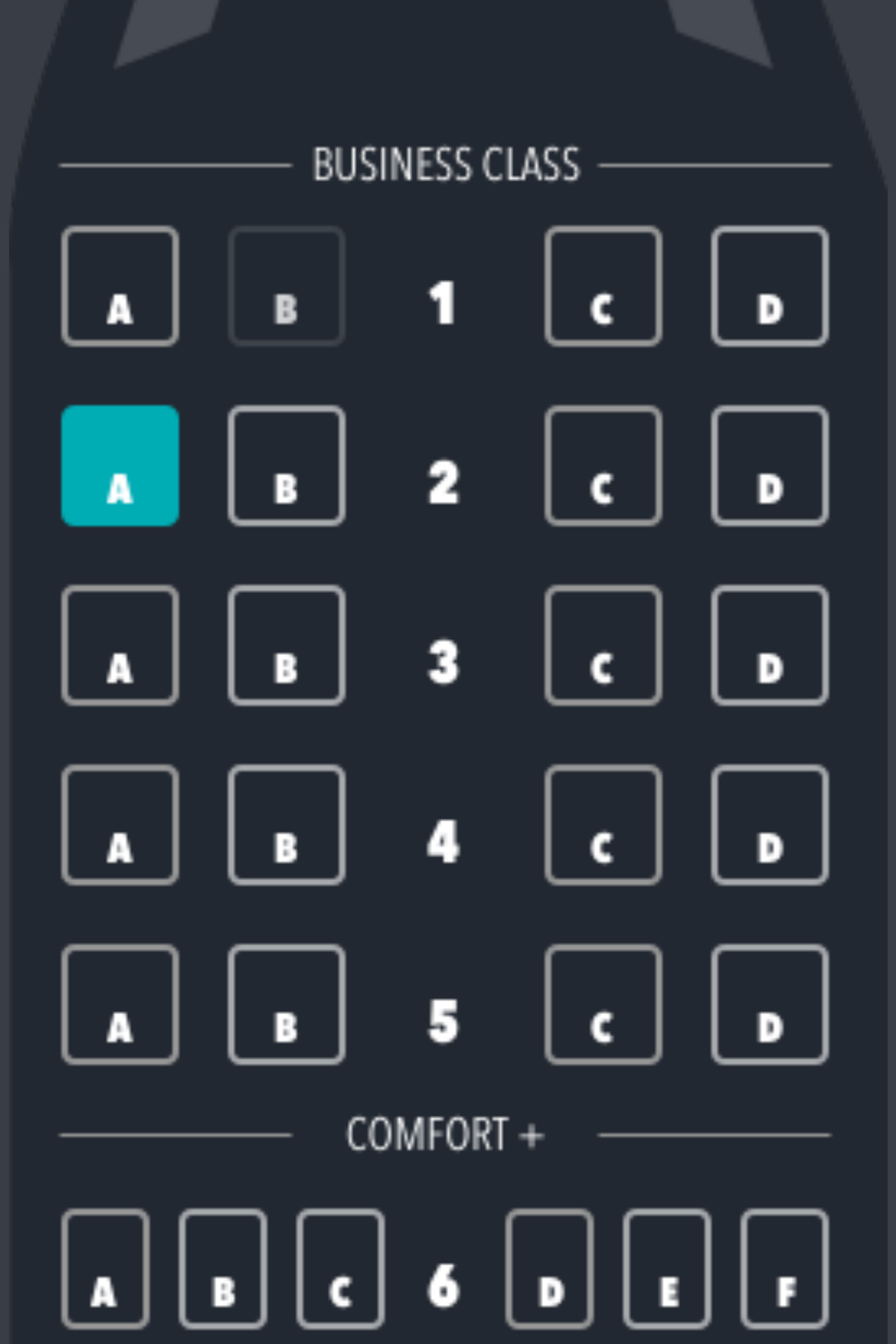
# SCENARIO 1:
# Selecting an Available Seat

- **Given:** The user is viewing an airplane seating chart.
- **When:** The user selects an available first-class seat "2A".

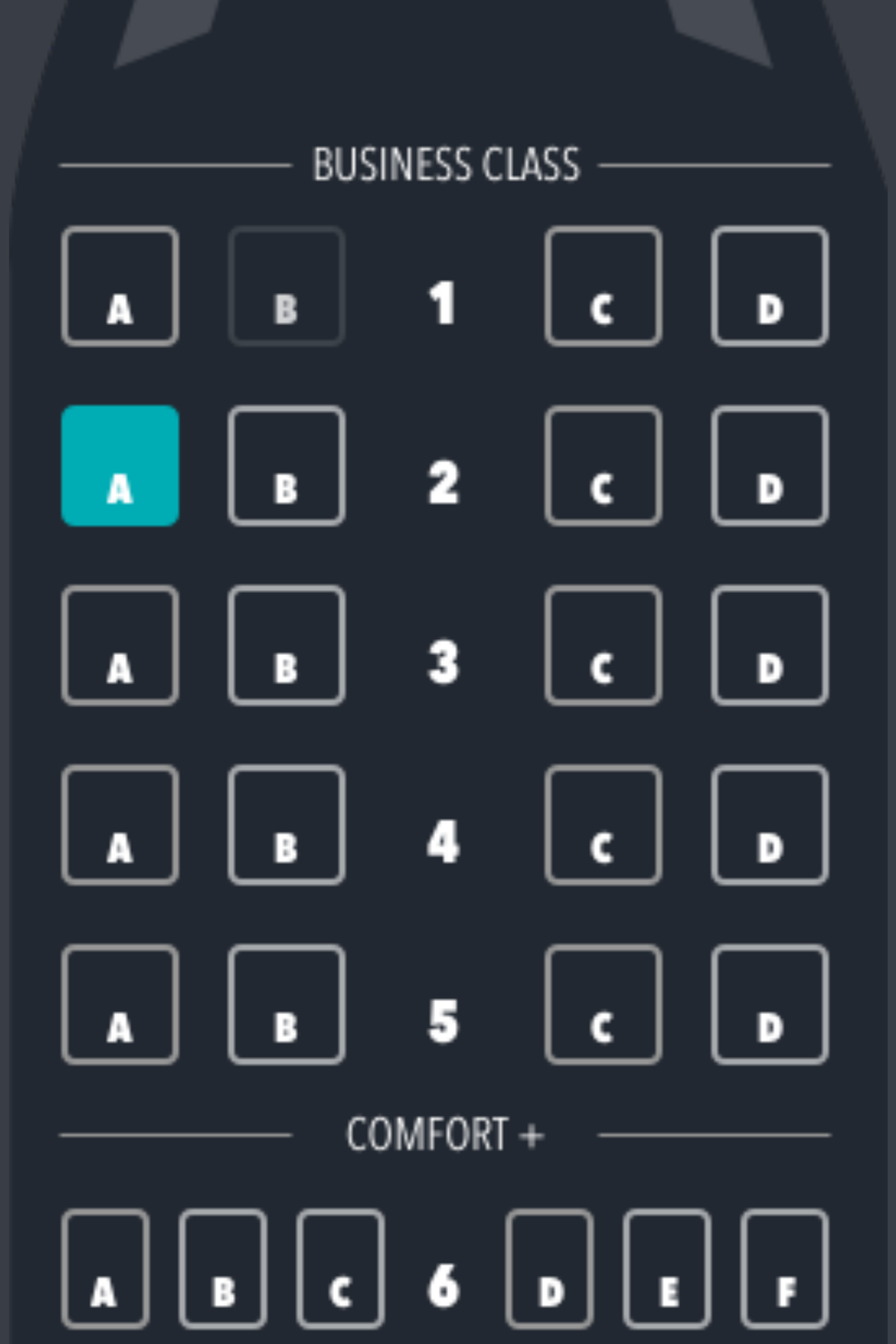


# SCENARIO 1:
# Selecting an Available Seat

- **Given:** The user is viewing an airplane seating chart.
- **When:** The user selects an available first-class seat "2A".
- **Then:** The  seat should be marked as selected, and a confirmation should indicate that the user has successfully selected the seat.

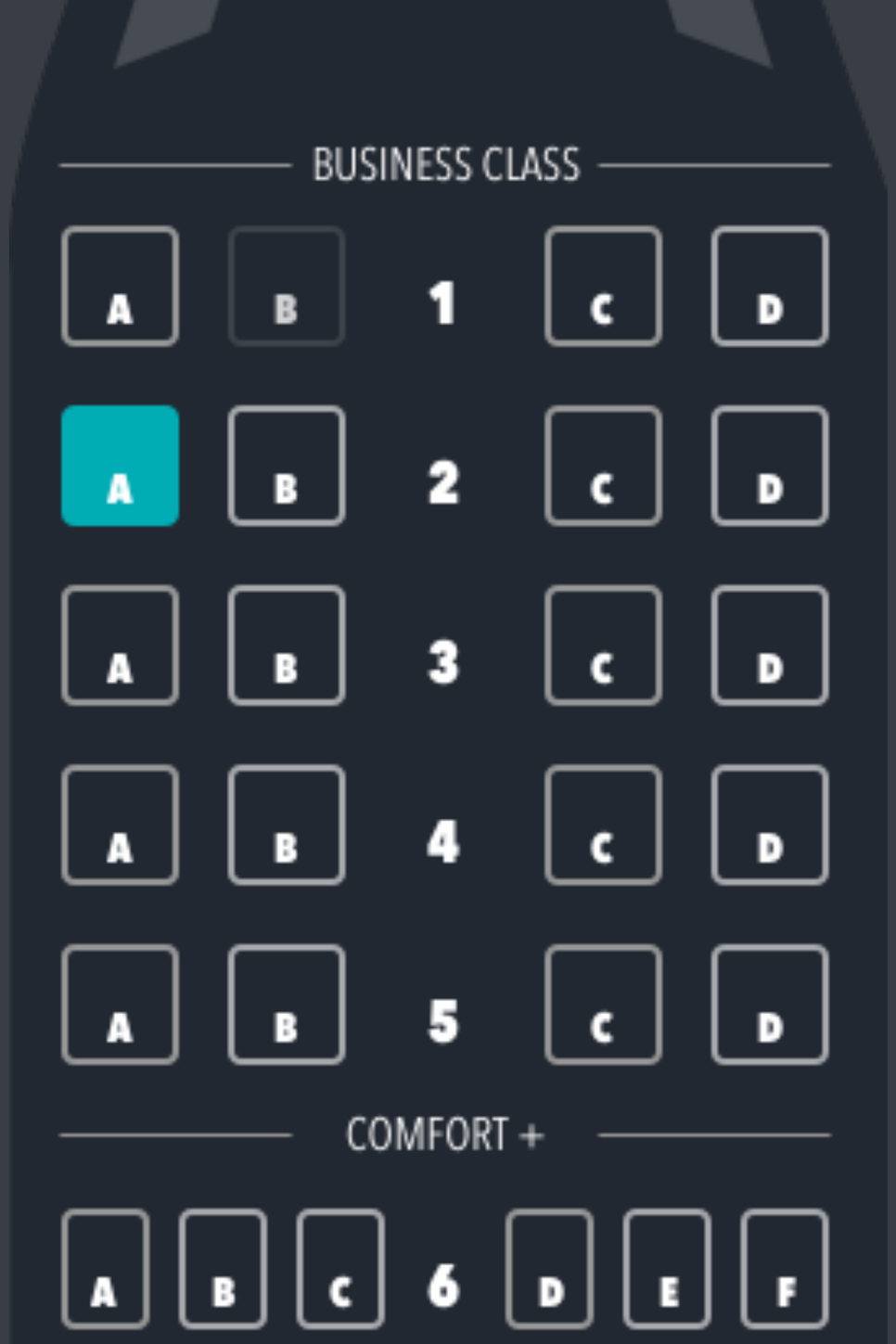


# SCENARIO 2:
# Preventing Unavailable Seat Selection



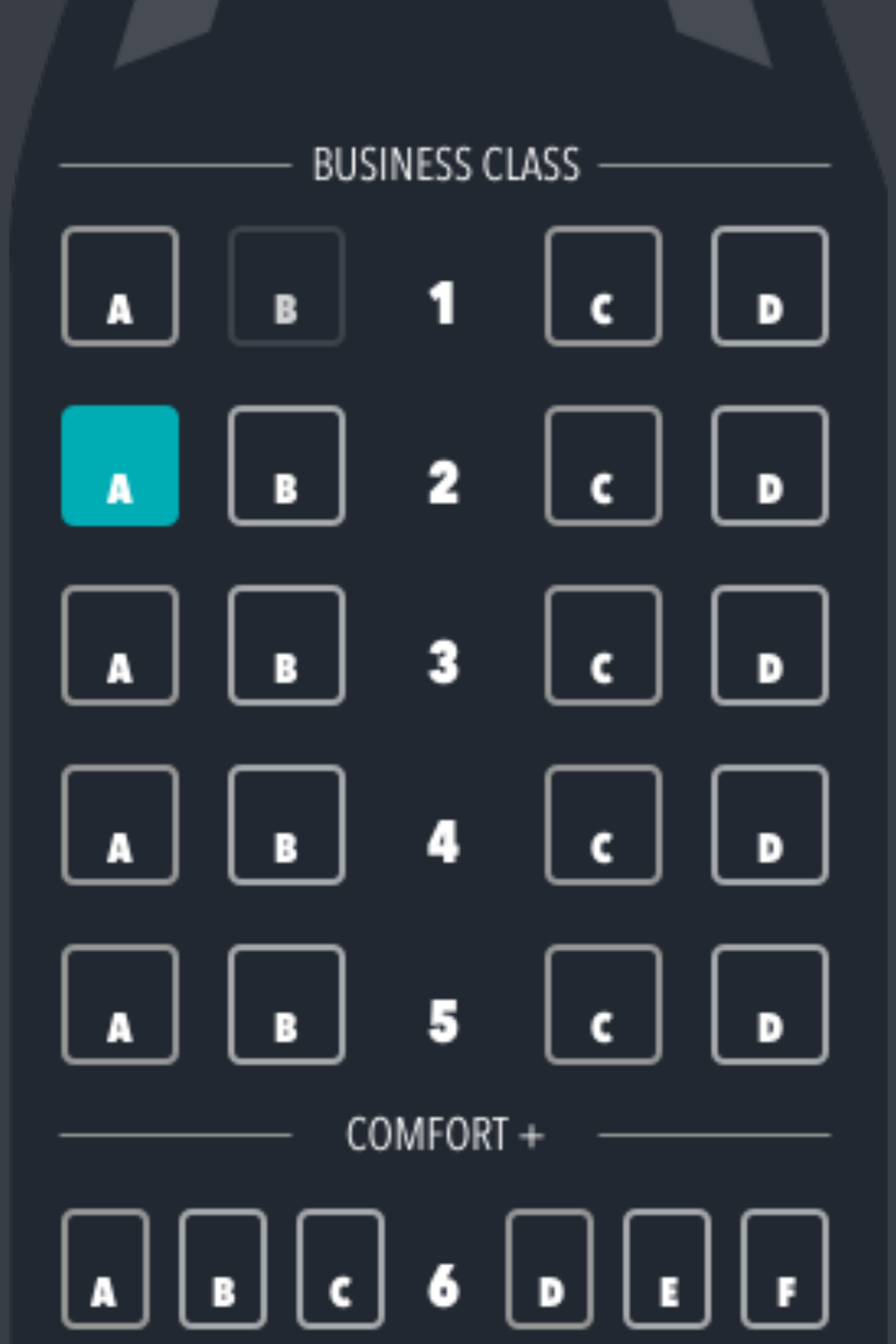
# SCENARIO 2:
# Preventing Unavailable Seat Selection

- **Given:** An unavailable first-class seat "1B" is shown on the chart



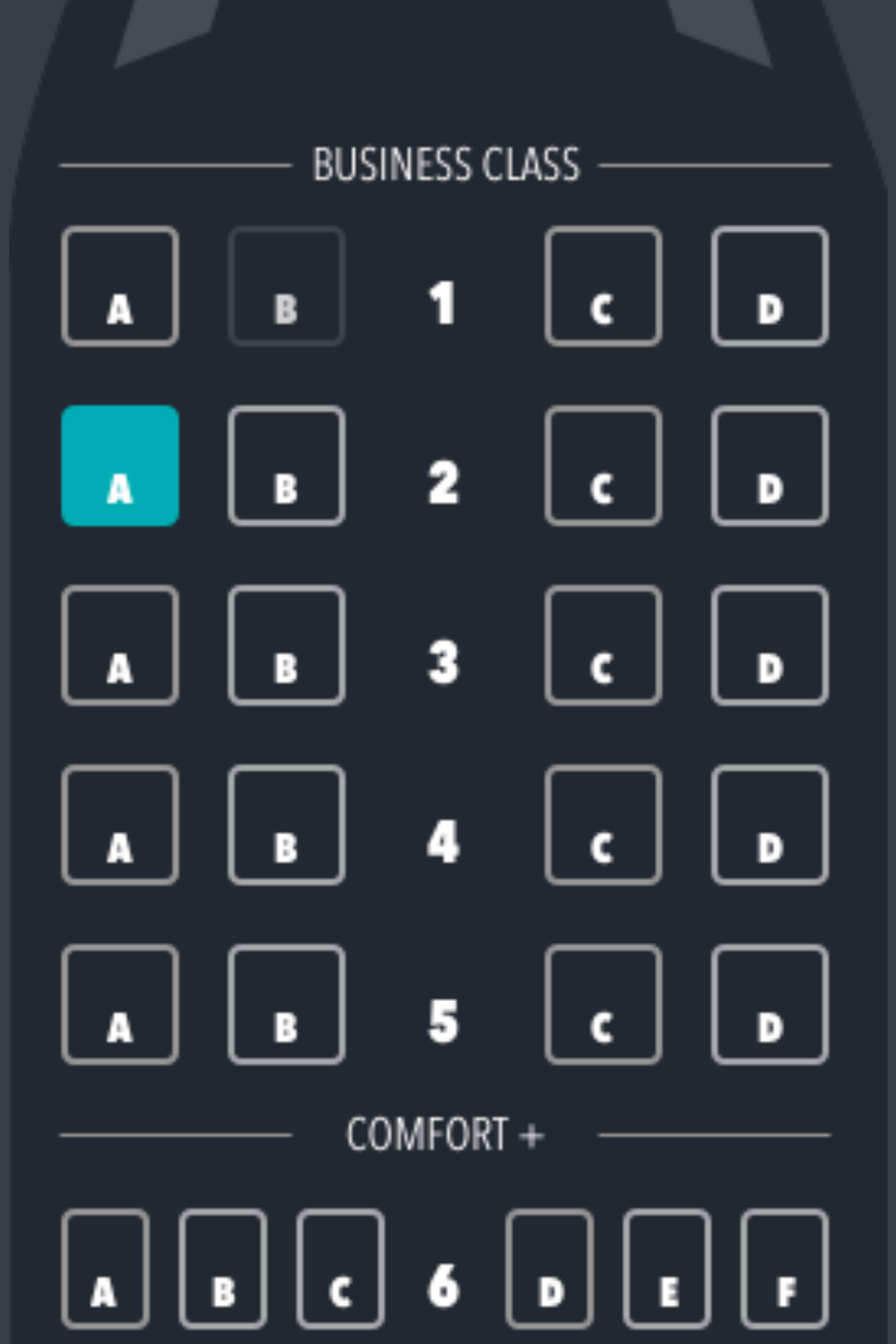
## SCENARIO 2:
# Preventing Unavailable Seat Selection

- **Given:** An unavailable first-class seat "1B" is shown on the chart
- **When:** The user tries to select seat "1B".

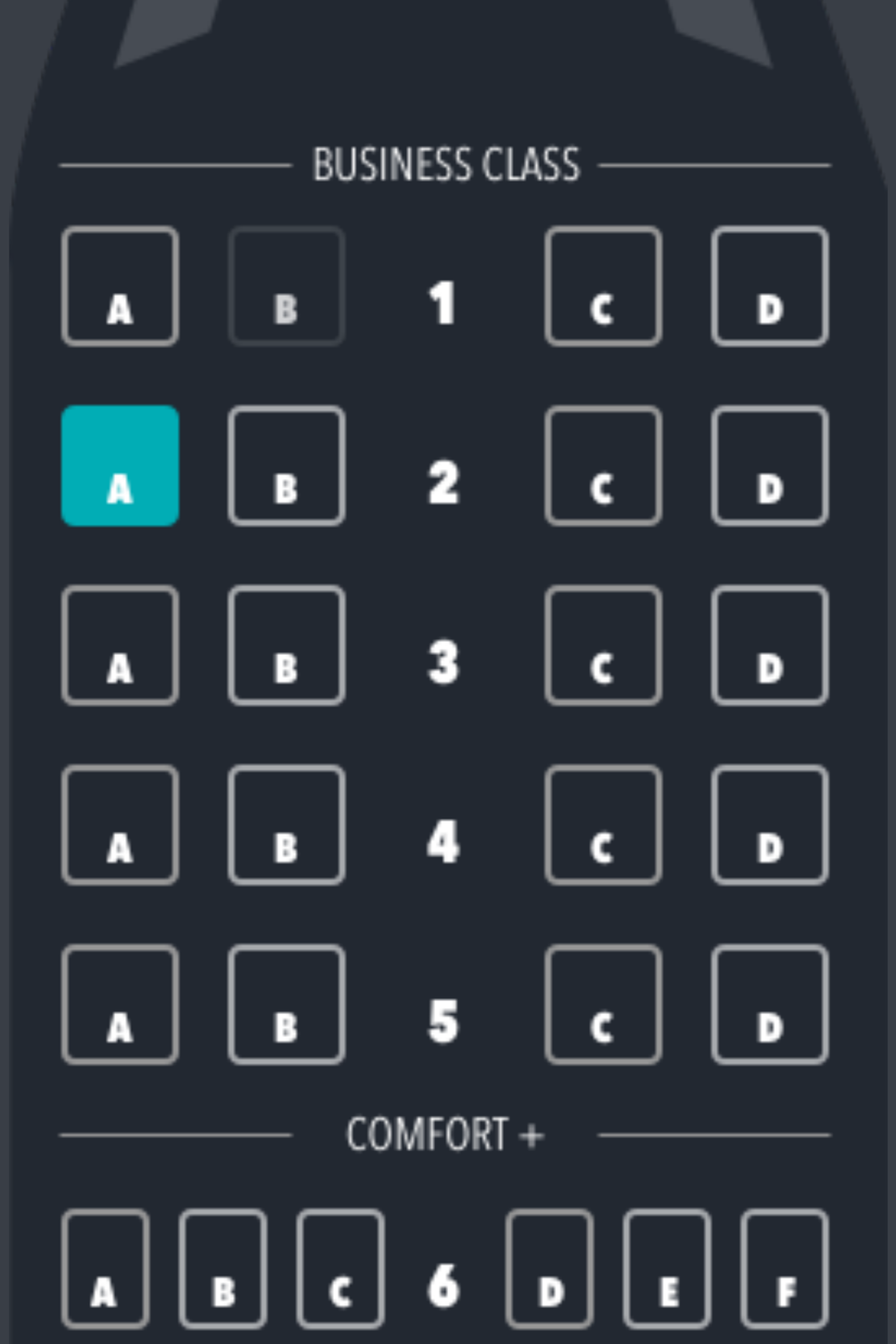


## SCENARIO 2:
# Preventing Unavailable Seat Selection

- **Given:** An unavailable first-class seat "1B" is shown on the chart
- **When:** The user tries to select seat "1B".
- **Then:** The selection should fail, and the user should receive an indication that the seat is unavailable.

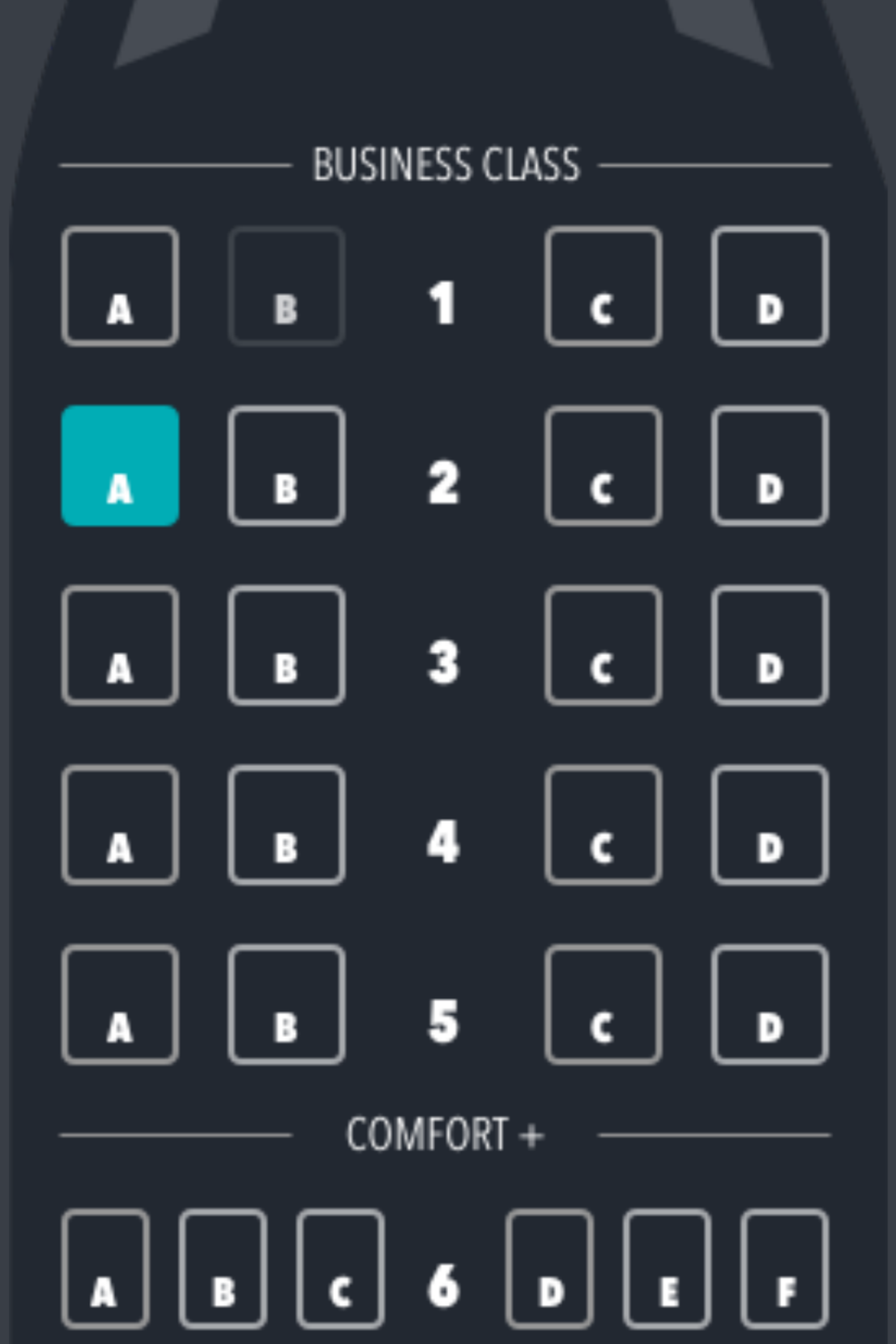


# SCENARIO 3:
# Restricting Seat Selection to a Specific Class



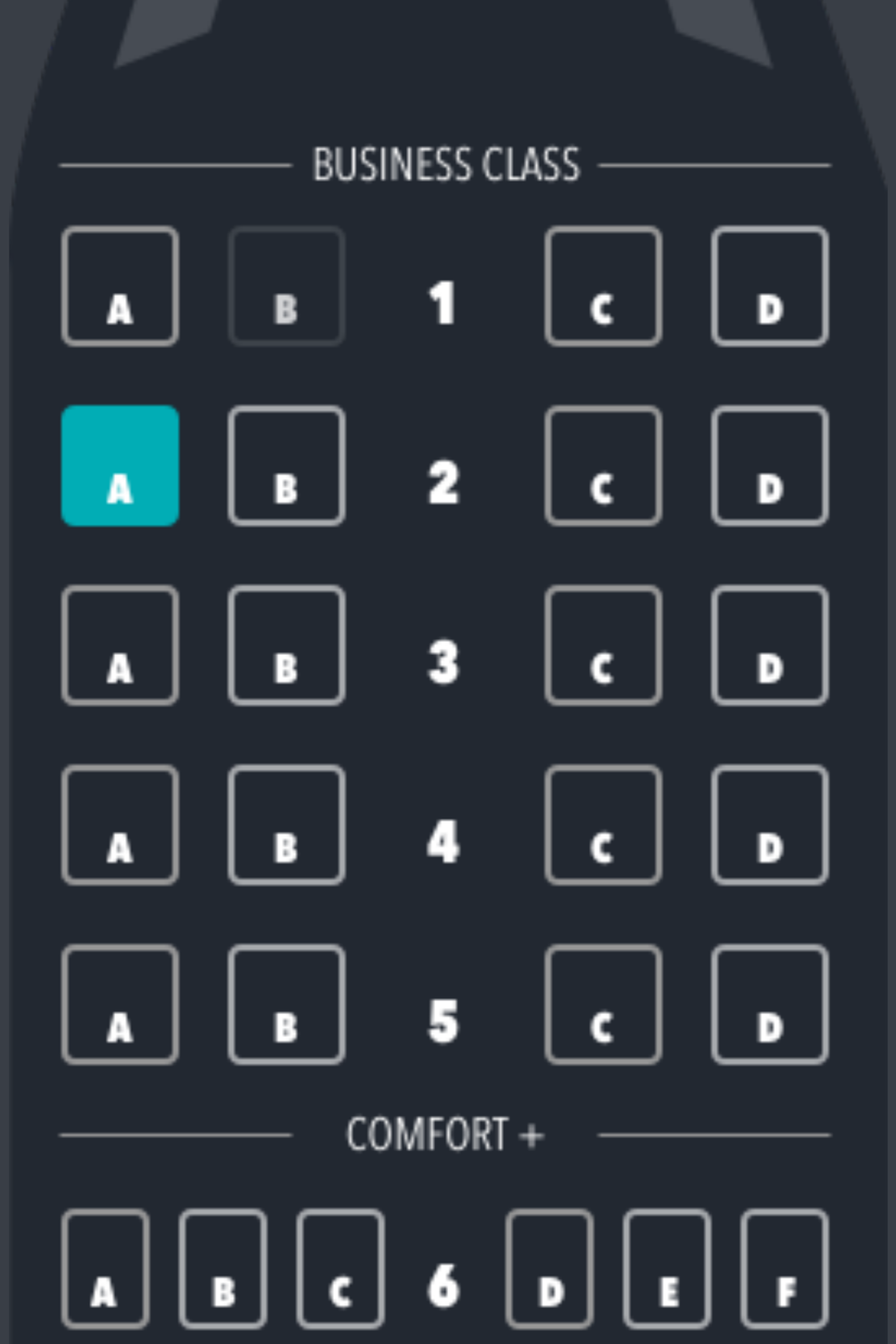
## SCENARIO 3:
# Restricting Seat Selection to a Specific Class

- **Given:** The user has booked a ticket in the economy class.



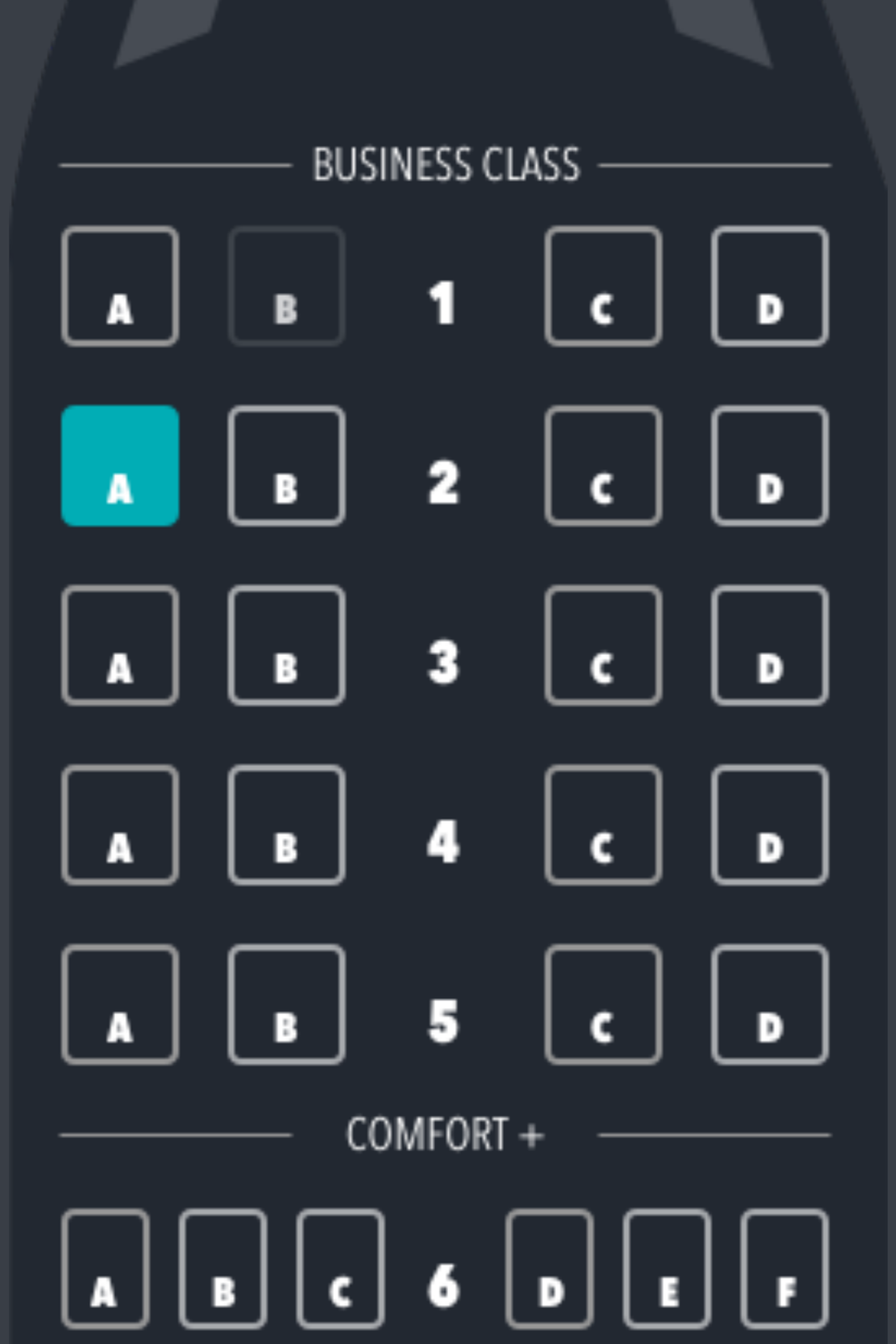
## SCENARIO 3:
# Restricting Seat Selection to a Specific Class

- **Given:** The user has booked a ticket in the economy class.
- **When:** The user tries to select a first-class seat "1A".



## SCENARIO 3:
# Restricting Seat Selection to a Specific Class

- **Given:** The user has booked a ticket in the economy class.
- **When:** The user tries to select a first-class seat "1A".
- **Then:** The selection should fail, and a message should inform the user that only economy class seats can be selected.



# PRACTICAL EXAMPLE (CONT.)

STEP FOUR

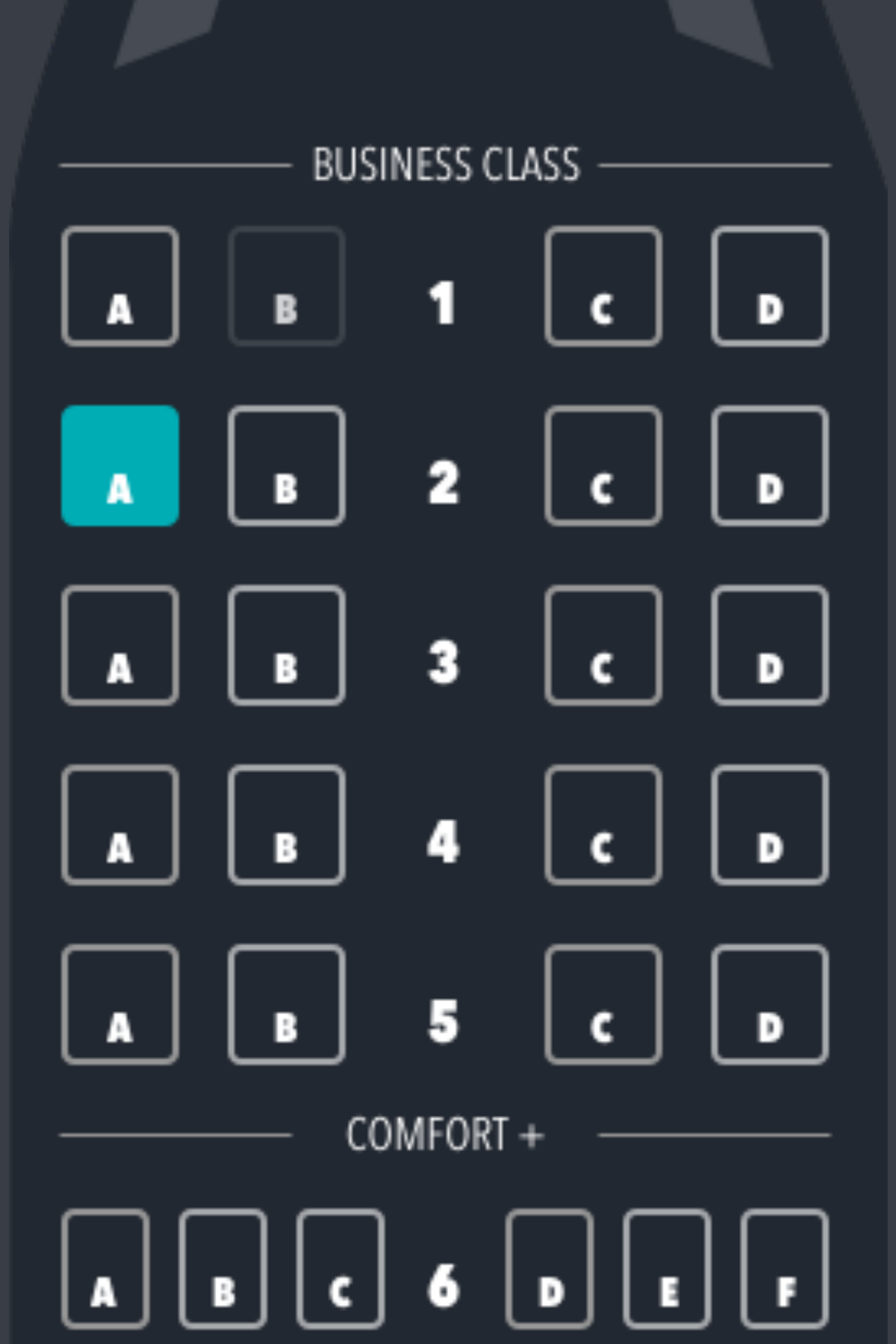
# WRITE UNIT TESTS

USING MOCKS & STUBS

# SCENARIO 1

## STEP 1:
# Selecting an Available Seat

- **Given:** The user is viewing an airplane seating chart.
- **When:** The user selects an available first-class seat "2A".
- **Then:** The  seat should be marked as selected, and a confirmation should indicate that the user has successfully selected the seat.



```swift
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat()
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub()
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub(seats: [seat])
}
```

```swift
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected()
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)

    // Then
    XCTAssertTrue()
}
```

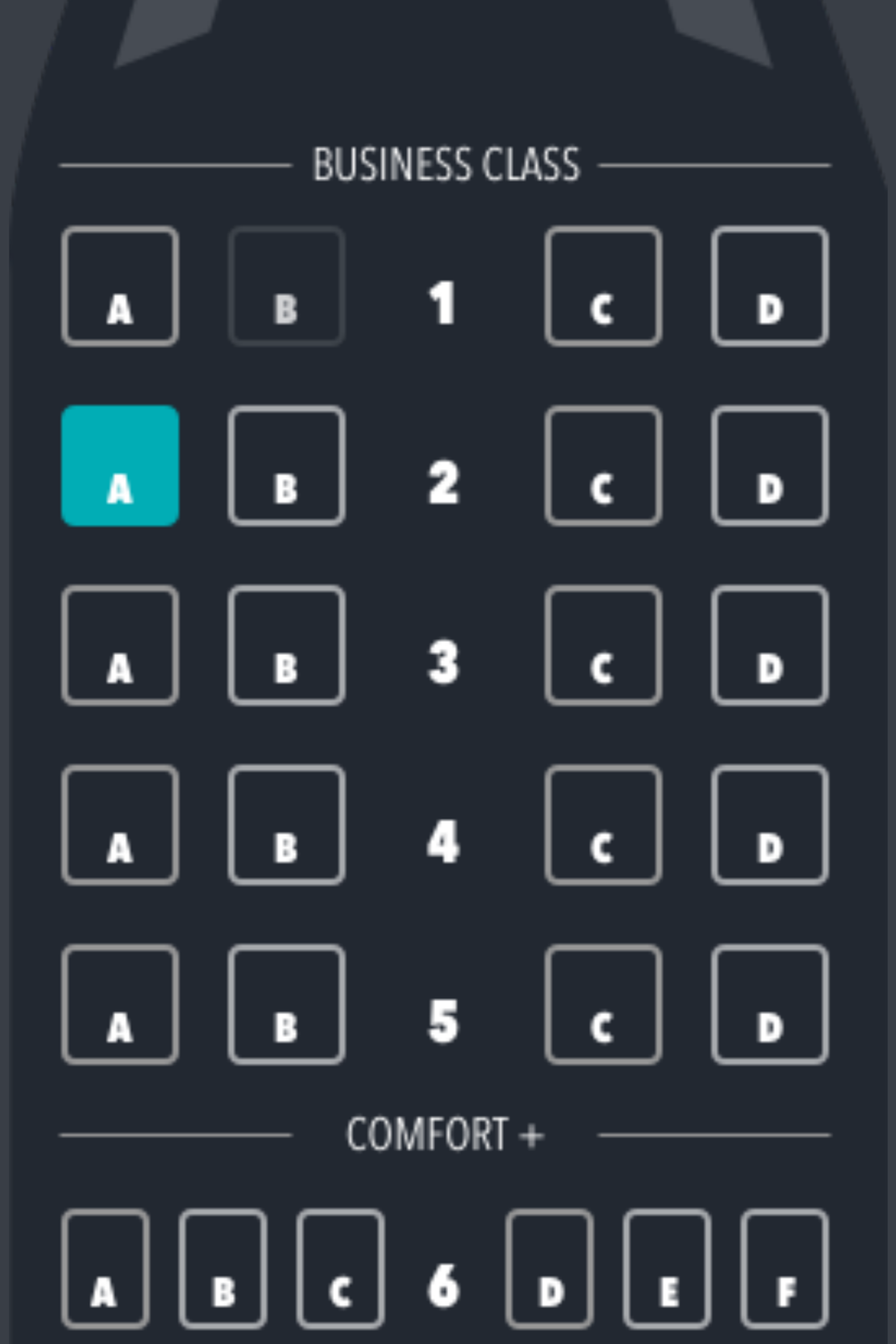
```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)

    // Then
    XCTAssertTrue(manager.seats.first!.isSelected)
}
```

```
//Feature: Seat Selection Management
func testToggleSelectedSeat_WhenSeatIsAvailable_ShouldMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: true, isSelected: false,
                        classType: .economy)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)

    // Then
    XCTAssertTrue(manager.seats.first!.isSelected,
                  "Available seat should be marked as selected")
}
```

# SCENARIO 2

## STEP 2:
# Preventing Unavailable Seat Selection

- **Given:** An unavailable first-class seat "1B" is shown on the chart
- **When:** The user tries to select seat "1B".
- **Then:** The selection should fail, and the user should receive an indication that the seat is unavailable.



```swift
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat()
}
```

```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(id: UUID(), isAvailable: false,
                        isSelected: false, classType: .economy)
}
```

```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
}
```

```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
    let manager = SeatDataManagerStub(seats: [seat])
}
```

```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)
}
```

```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)

    // Then
    XCTAssertFalse()
}
```

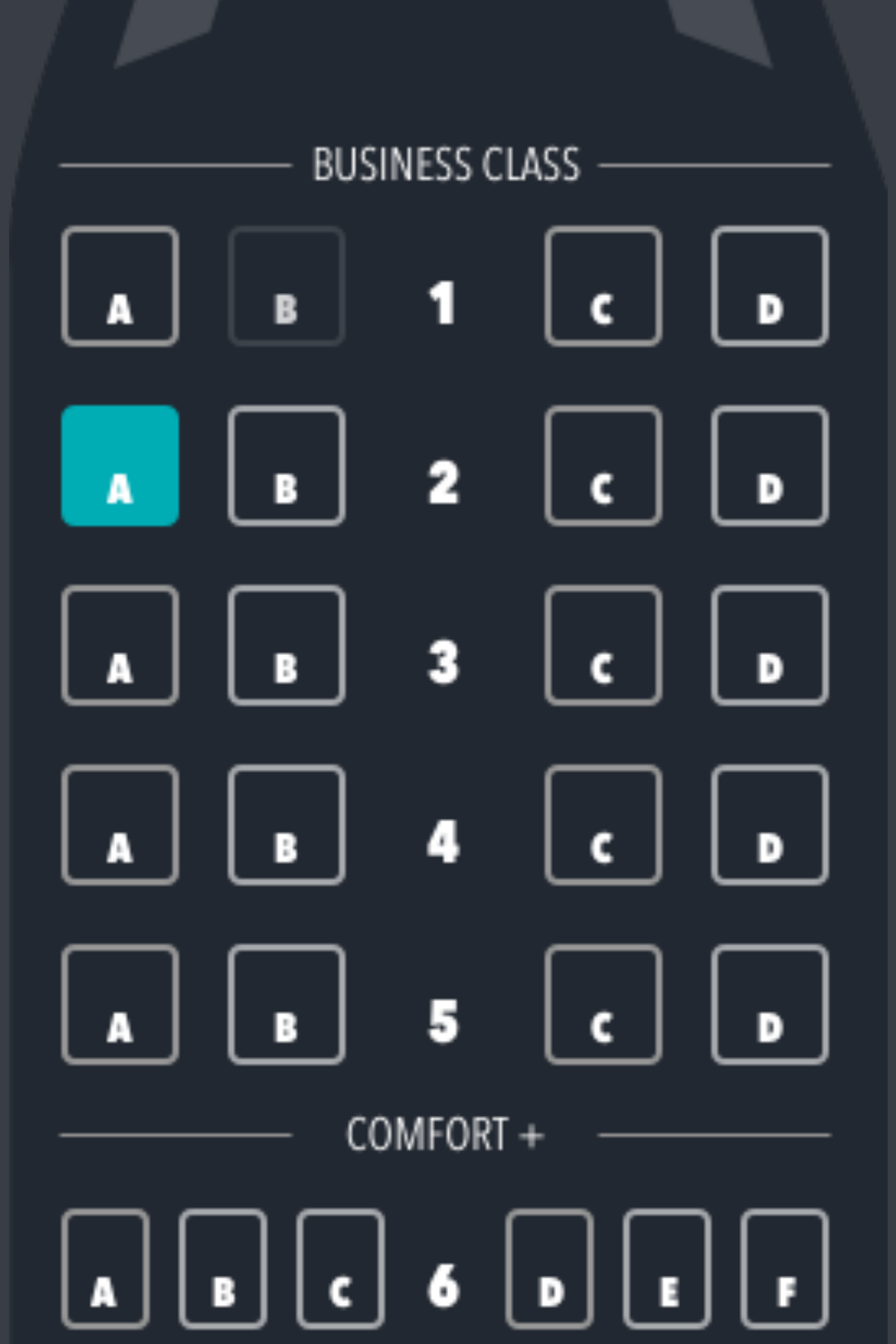
```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)

    // Then
    XCTAssertFalse(manager.seats.first!.isSelected)
}
```

```
func testToggleSelectedSeat_WhenSeatIsUnavailable_ShouldNotMarkSeatAsSelected() {
    // Given
    let seat = MockSeat(•••)
    let manager = SeatDataManagerStub(seats: [seat])

    // When
    manager.toggleSelected(seat)

    // Then
    XCTAssertFalse(manager.seats.first!.isSelected, "Unavailable seat should not
                                                     be marked as selected")
}
```

# SCENARIO 3

## STEP 3:
# Restricting Seat Selection to a Specific Class

- **Given:** The user has booked a ticket in the economy class.
- **When:** The user tries to select a first-class seat "1A".
- **Then:** The selection should fail, and a message should inform the user that only economy class seats can be selected.



```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [MockSeat(•••),
                        MockSeat(•••)]
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [MockSeat(•••)]
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]

}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub()
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])
}
```

```swift
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
    firstClassSeats = manager.seats.filter { $0.classType == .first }
}
```

```
func testSeatSelection_whenSelectingSeatsWithDifferentTypes_checkRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
    firstClassSeats = manager.seats.filter { $0.classType == .first }
}
```

```
func testSeatSelection_whenselectingseatswithDifferentTypes_checkRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
    firstClassSeats = manager.seats.filter { $0.classType == .first }

    // Then
}
```

```
func testSeatSelection_whenSelectingSeatsWithDifferentTypes_checkRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
    firstClassSeats = manager.seats.filter { $0.classType == .first }

    // Then
    XCTAssertTrue(economySeats[0].isSelected, "1st economy seat should be selected.")
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
    firstClassSeats = manager.seats.filter { $0.classType == .first }

    // Then
    XCTAssertTrue(economySeats[0].isSelected, "1st economy seat should be selected.")
    XCTAssertFalse(economySeats[1].isSelected, "2nd economy seat shouldn’t be selected.")
}
```

```
func testSeatSelection_WhenSelectingSeatsWithDifferentTypes_ShouldRespectTypeLimits() {
    // Given
    var economySeats = [•••]
    var firstClassSeats = [•••]
    let seats = economySeats + firstClassSeats
    let manager = SeatDataManagerStub(seats: seats,
                                      selectionLimitByType: [.economy: 1, .first: 1])

    // When
    manager.toggleSelected(economySeats[0])
    manager.toggleSelected(economySeats[1])
    manager.toggleSelected(firstClassSeats[0])

    economySeats = manager.seats.filter { $0.classType == .economy }
    firstClassSeats = manager.seats.filter { $0.classType == .first }

    // Then
    XCTAssertTrue(economySeats[0].isSelected, "1st economy seat should be selected.")
    XCTAssertFalse(economySeats[1].isSelected, "2nd economy seat shouldn’t be selected.")
    XCTAssertTrue(firstClassSeats[0].isSelected, "1st class seat should be selected.")
}
```

STEP FIVE

# DEVELOP

INTEGRATION TESTS

STEP SIX

# REFINE & ITERATE

# DS {to}

# WHAT WE LEARNED

5

# WHAT WE COVERED

- The backstory
- Inside vs Outside
- User stories
- Practical example
- Wrapping up

# Design-driven DEVELOPMENT

# Q&A

by Craig Clayton

/designtoswiftui

thedevme

mastodon.cloud/@thedevme

DS {to}

THANK YOU!!