# CHESS ONLINE

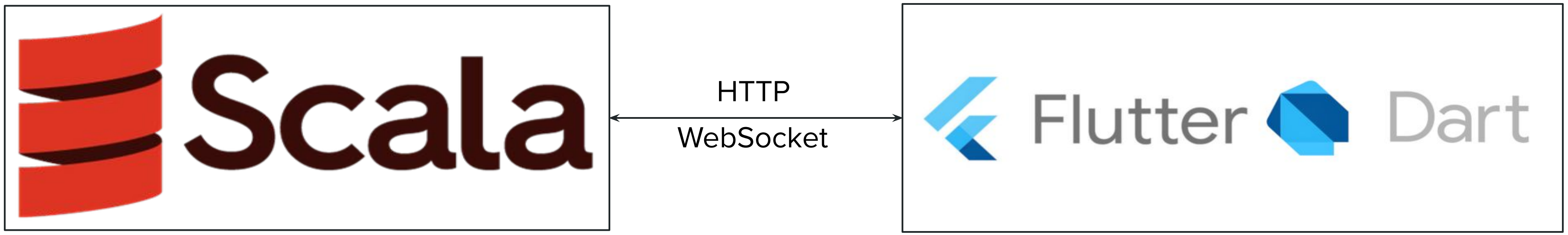
By Daniyil Yevtyushkin

| | |
|---|---|
| Project's Github: | github.com/yevtyushkin/chess_online |
| Deployed web app: | scala-chess-online.web.app |

## Motivation

- Domain and rules are complex and challenging
- Multiplayer implementation touches most of the taught Scala Bootcamp topics
- Chess is one of my hobbies

## Project Components



**Backend**:
- Chess domain and rules
- Multiplayer:
  - Player creation and authorization
  - Room creation and establishing connections
  - Handling game

**Frontend**:
- Login page
- Rooms page
- Game and Game Results page
- Supports multiple platforms:
  - Web **(was the main target)**
  - Mobile (iOS, Android)
  - Desktop

## Libraries used

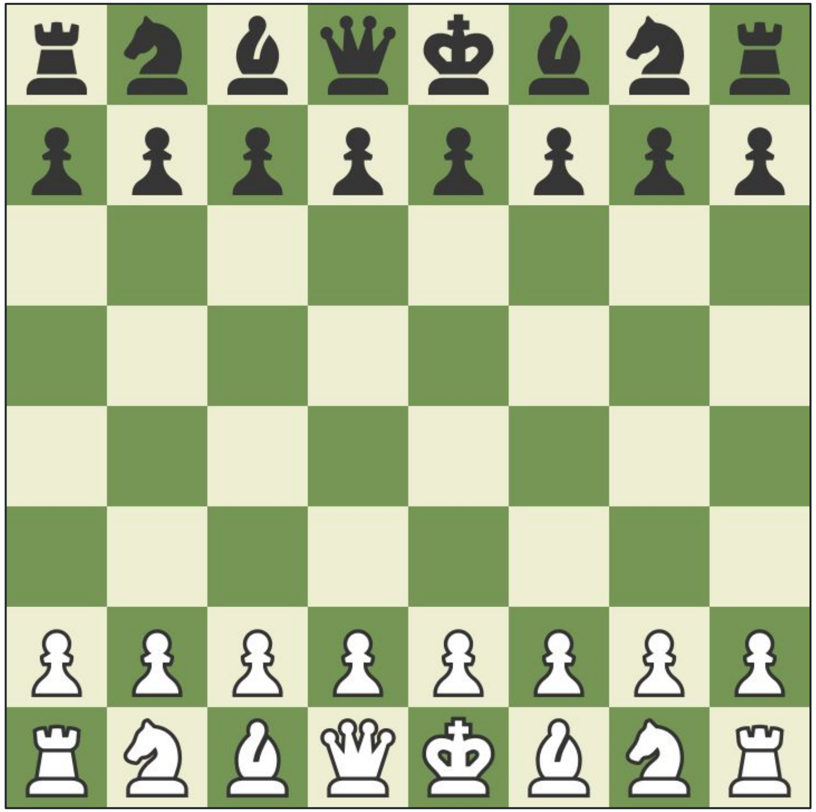
- enumeratum
- cats
- cats-effect
- http4s
- fs2
- circe
- scalatest + scalamock

## Basic chess rules

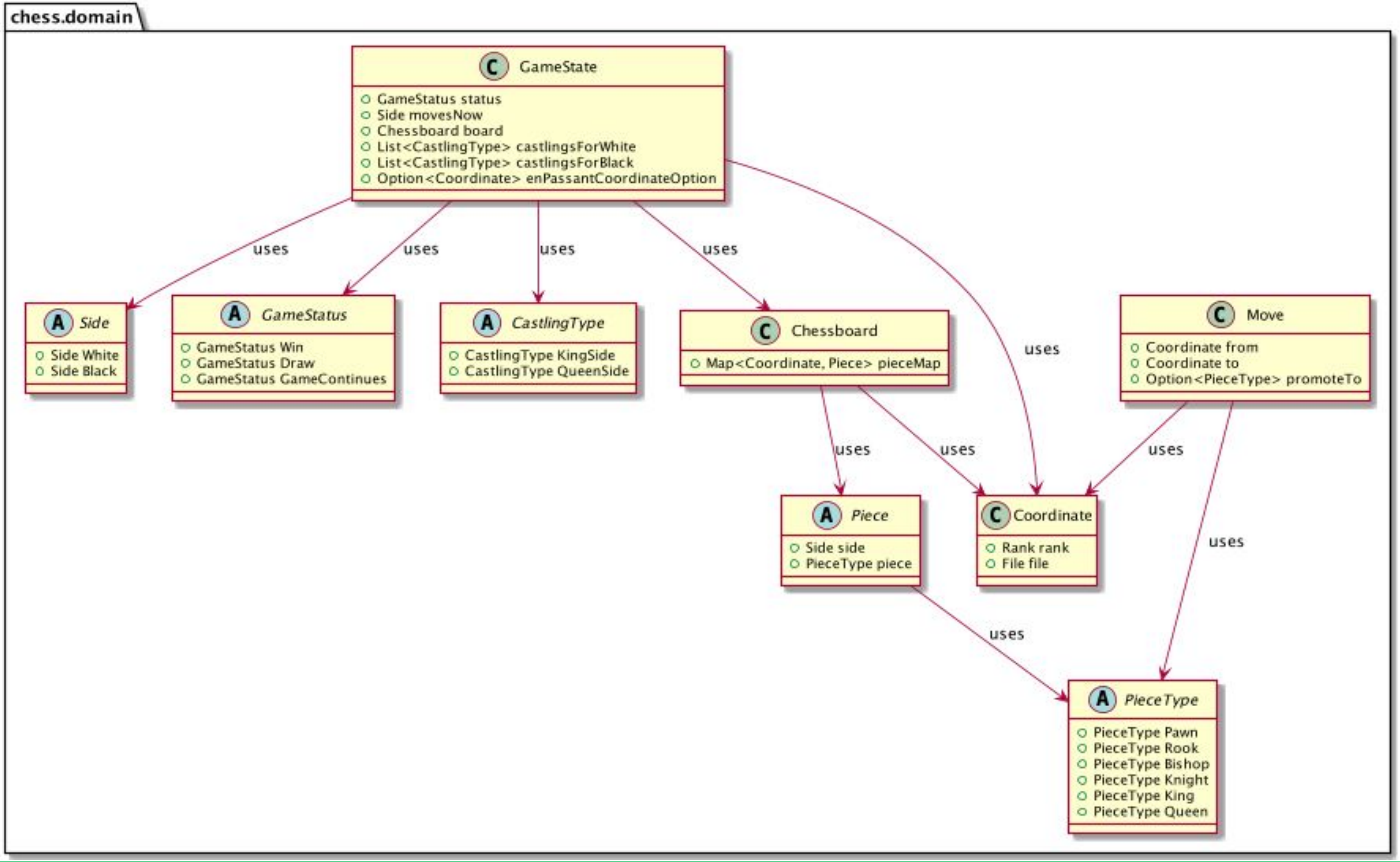
- Two sides: White & Black
- 16 pieces for each side of different types
- The goal is to attack the opponent’s king so it can’t escape (checkmate)

## What’s implemented

- Basic moves and captures
- Castlings
- Pawn promotions
- Handling En-Passant squares
- Game result determination (Checkmate / Draw)
- Complete validation for moves

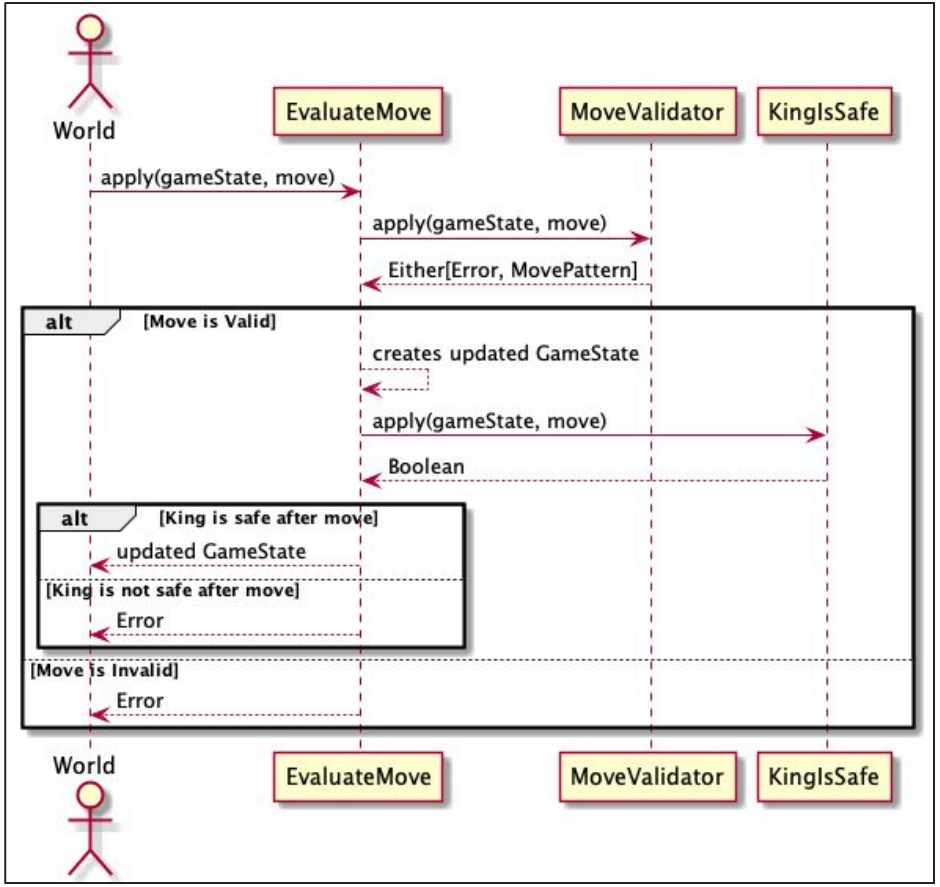

Initial chess board setup

# Domain Modeling



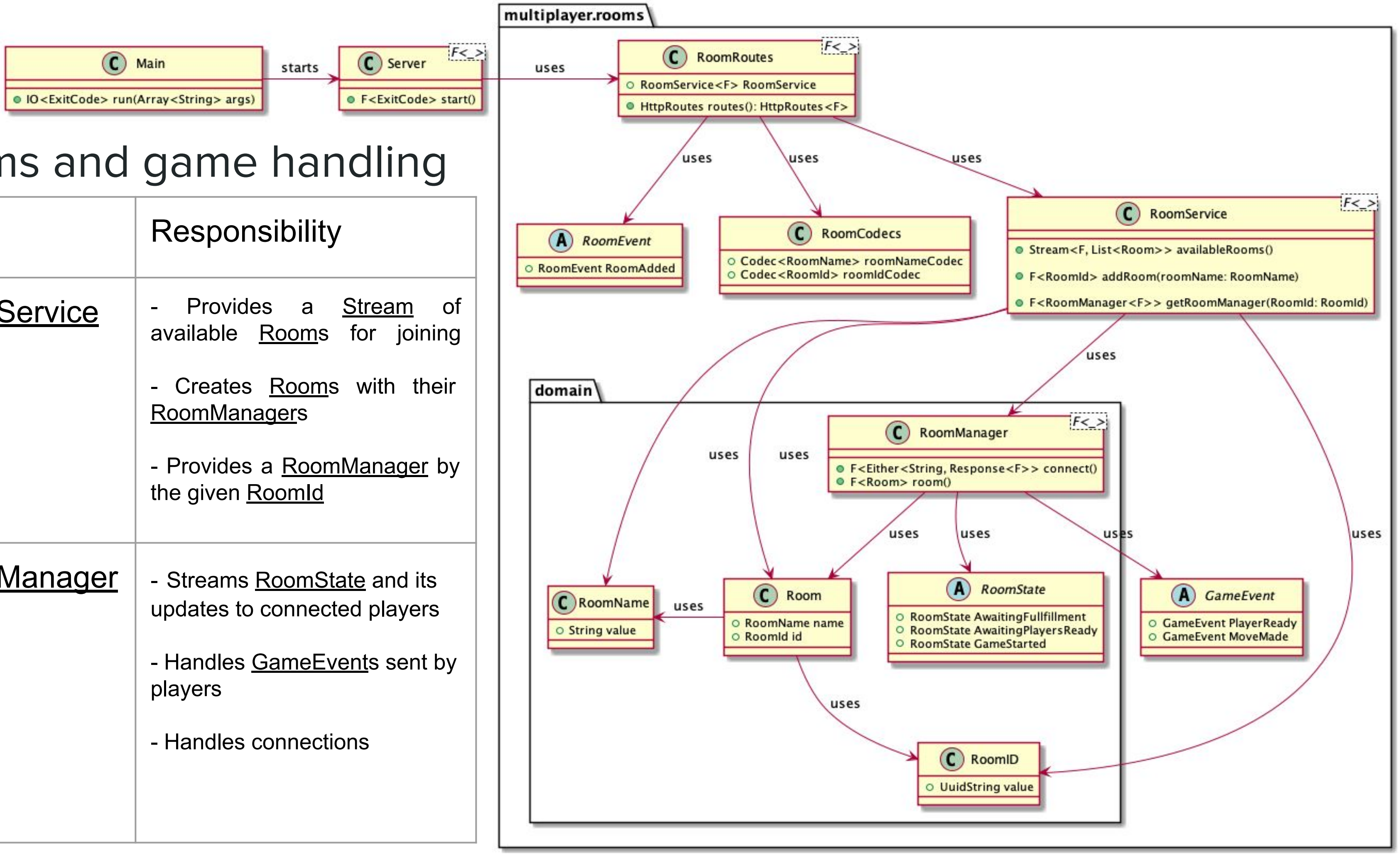
## Move validation and evaluation

| Trait | Responsibility |
|---|---|
| EvaluateMove | Validates and evaluates the given Move. Returns an updated GameState or an error |
| MoveValidator | Validates the given Move. Returns a MovePattern or an error |
| KingIsSafe | Checks if the King has a safe position. Returns a Boolean |



## Rooms and game handling

| Trait | Responsibility |
|---|---|
| RoomService | - Provides a Stream of available Rooms for joining<br><br>- Creates Rooms with their RoomManagers<br><br>- Provides a RoomManager by the given RoomId |
| RoomManager | - Streams RoomState and its updates to connected players<br><br>- Handles GameEvents sent by players<br><br>- Handles connections |



Main
IO<ExitCode> run(Array<String> args)
starts
Server
F<_>
F<ExitCode> start()
uses
multiplayer.rooms
RoomRoutes
F<_>
RoomService<F> RoomService
HttpRoutes routes(): HttpRoutes<F>
uses
uses
uses
RoomEvent
RoomEvent RoomAdded
RoomCodecs
Codec<RoomName> roomNameCodec
Codec<RoomId> roomIdCodec
RoomService
F<_>
Stream<F, List<Room>> availableRooms()
F<RoomId> addRoom(roomName: RoomName)
F<RoomManager<F>> getRoomManager(RoomId: RoomId)
uses
uses
uses
uses
domain
RoomManager
F<_>
F<Either<String, Response<F>> connect()
F<Room> room()
uses
uses
uses
RoomName
String value
uses
Room
RoomName name
RoomId id
RoomState
RoomState AwaitingFullfillment
RoomState AwaitingPlayersReady
RoomState GameStarted
GameEvent
GameEvent PlayerReady
GameEvent MoveMade
uses
RoomID
UuidString value

# Testing

Chess rules implementation has a complete unit test coverage:

- EvaluateMoveSpec.scala (main scenarios):
    - Returns a correct error if validation fails
    - Updates GameState correctly depending on the MovePattern returned by MoveValidator
    - Updates GameStatus correctly
- ValidateMoveSpec.scala (main scenarios):
    - Returns a correct error if the move is invalid
    - Returns the correct MovePattern if the move is valid (for different piece types
- KingIsSafeSpec.scala:
    - Returns true/false if the king can/can’t be attacked by an enemy piece
- And other domain classes’ helper methods

Used mocking for class dependencies

No unit tests for Multiplayer

Tests helped me a lot at early development stages

## CI/CD and Deployment

Service used: Github Actions

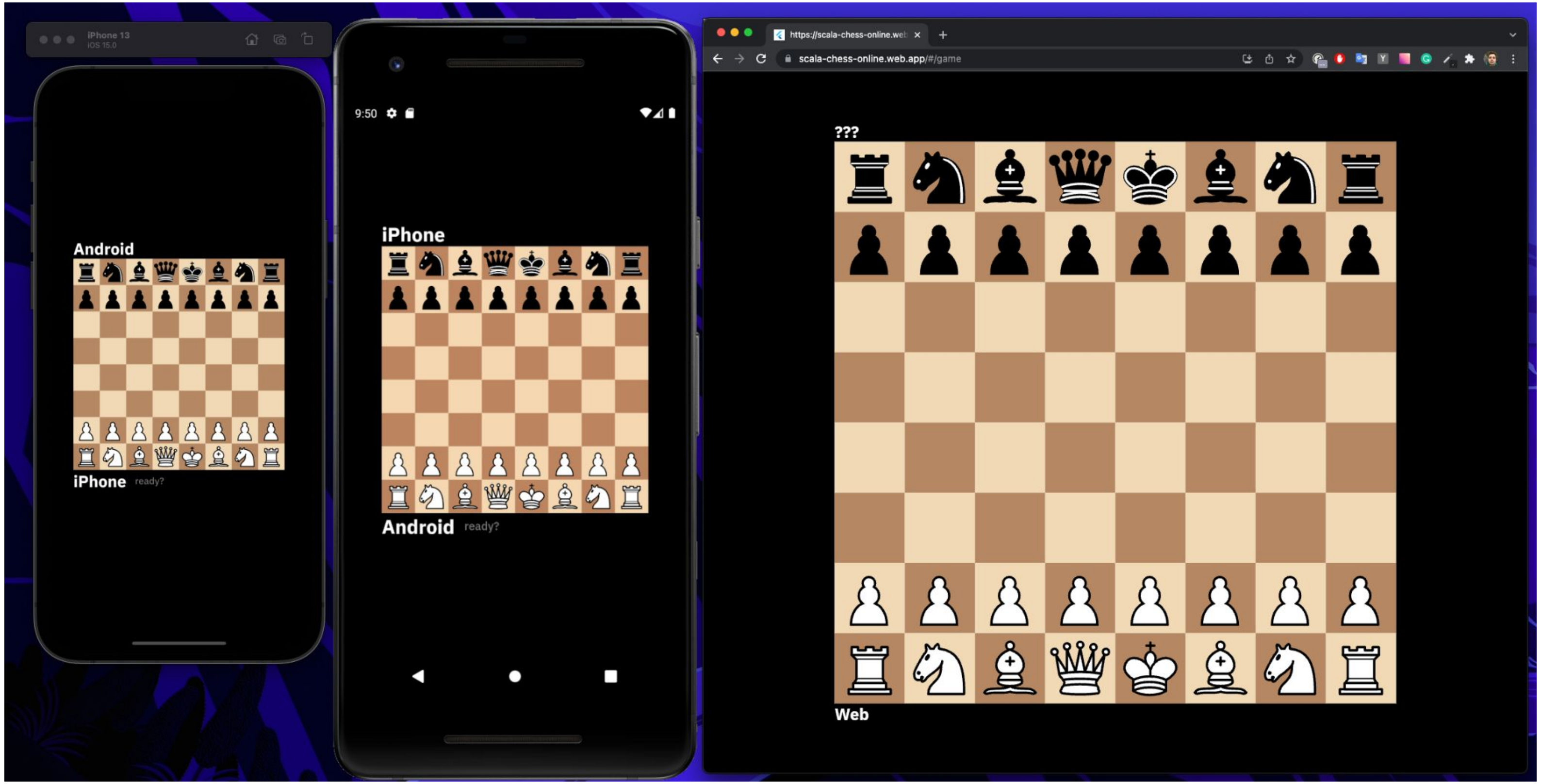
For Backend part:

- Running tests and submitting coverage to Codevoc
- Deployment to Heroku if the workflow succeeds

For Frontend part:

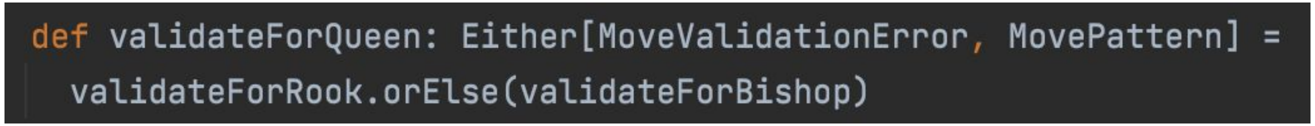
- Running code generation for generated models / codecs
- Compiling the app and its deployment to Firebase

## Frontend part



DEMO :)

## Interesting code examples

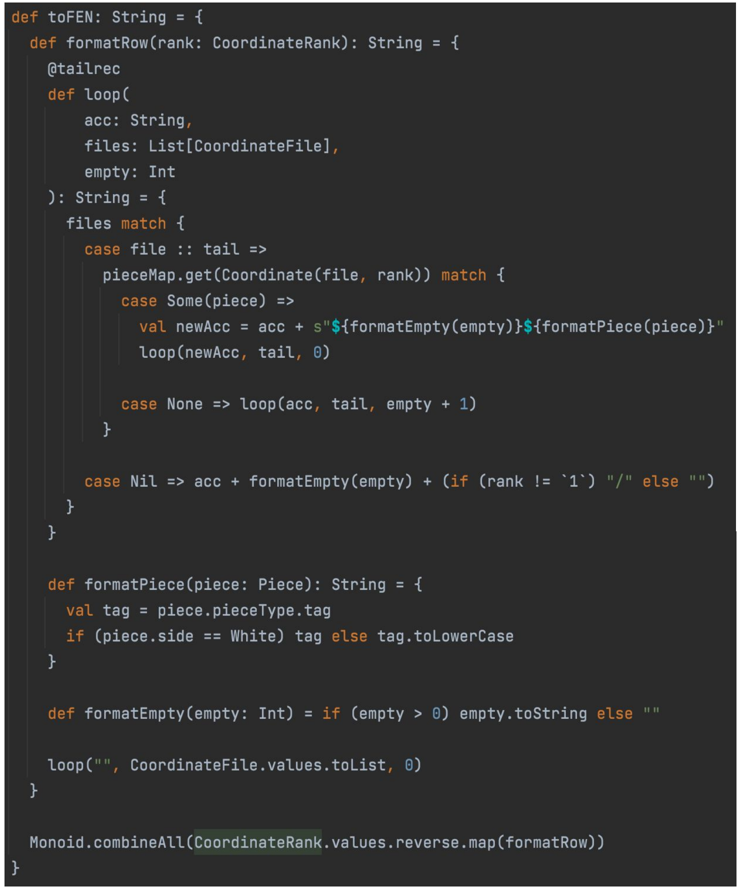


```scala
def validateForQueen: Either[MoveValidationError, MovePattern] =
  validateForRook.orElse(validateForBishop)
```

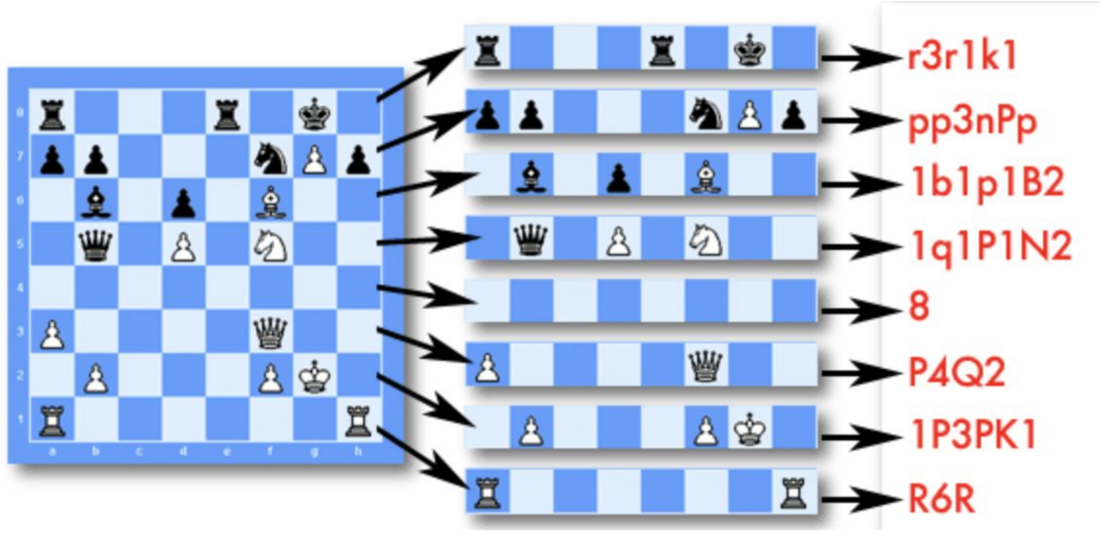
Queen move validation derived from rook and bishop validation

Why? DRY.

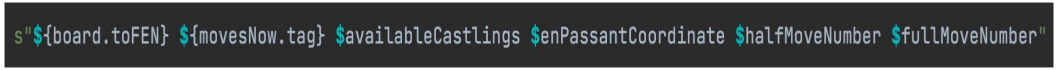
# Interesting code examples



```scala
def toFEN: String = {
  def formatRow(rank: CoordinateRank): String = {
    @tailrec
    def loop(
        acc: String,
        files: List[CoordinateFile],
        empty: Int
    ): String = {
      files match {
        case file :: tail =>
          pieceMap.get(Coordinate(file, rank)) match {
            case Some(piece) =>
              val newAcc = acc + s"${formatEmpty(empty)}${formatPiece(piece)}"
              loop(newAcc, tail, 0)

            case None => loop(acc, tail, empty + 1)
          }

        case Nil => acc + formatEmpty(empty) + (if (rank != `1`) "/" else "")
      }
    }

    def formatPiece(piece: Piece): String = {
      val tag = piece.pieceType.tag
      if (piece.side == White) tag else tag.toLowerCase
    }

    def formatEmpty(empty: Int) = if (empty > 0) empty.toString else ""

    loop("", CoordinateFile.values.toList, 0)
  }

  Monoid.combineAll(CoordinateRank.values.reverse.map(formatRow))
}
```



Converting piece positions to FEN notation



```scala
s"${board.toFEN} ${movesNow.tag} $availableCastlings $enPassantCoordinate $halfMoveNumber $fullMoveNumber"
```

Full FEN representation of GameState

Why? Compact JSON and compatibility with Frontend libraries.

## Future improvements and plans

- There’s no end of the game for the RoomManager, so it remains in memory after all players disconnect or the game ends

- Frontend can suddenly disconnect from the WebSocket connection, so reconnection feature for Frontend is a must

- Tests for Multiplayer

- Custom chess modes

## Conclusion

- I did a great project that met my expectations. The project was real fun and taught me how to do things in functional Scala.

- The Scala Bootcamp opened me a brand new world of functional programming with Scala. Thanks to all lecturers and organizers for their work and patience!

- Thanks to my mentor Ivan for all his help, knowledge and project suggestions that brought it to the next level!