The Quiplash Showdown

Go vs Scala

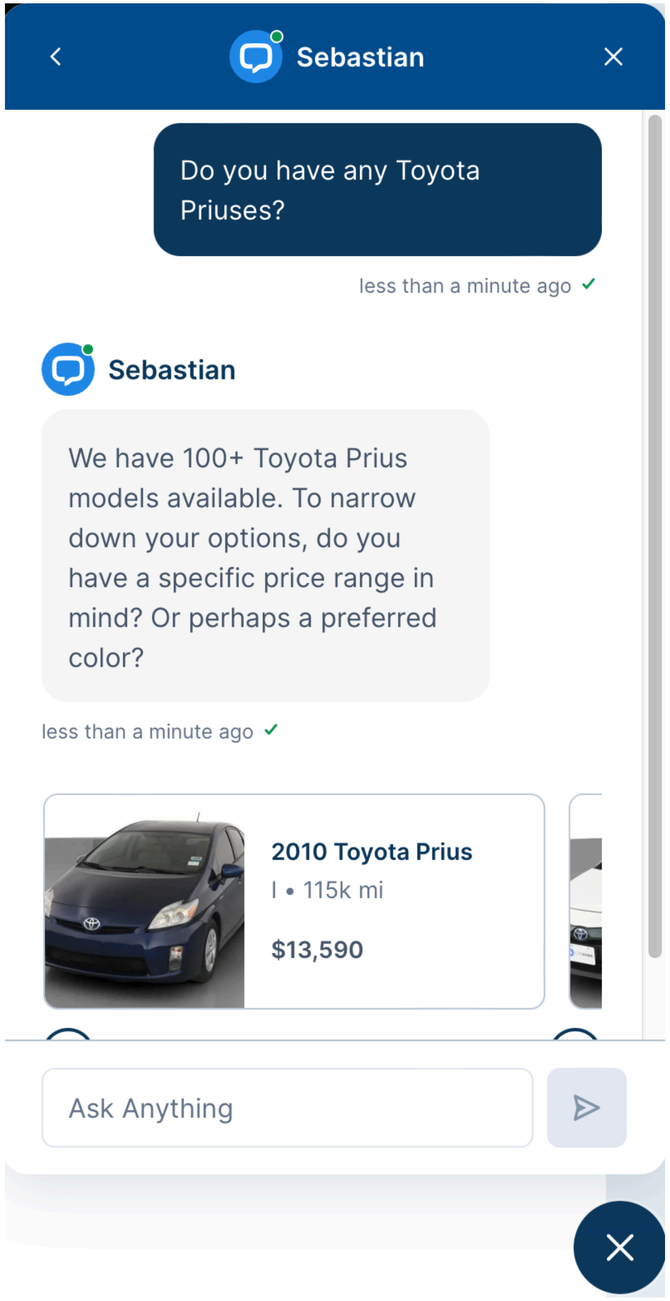
Lambdaconf 2024

About Me

Jim Weinert

- Backend developer
- Scala and Go Experience! YES!

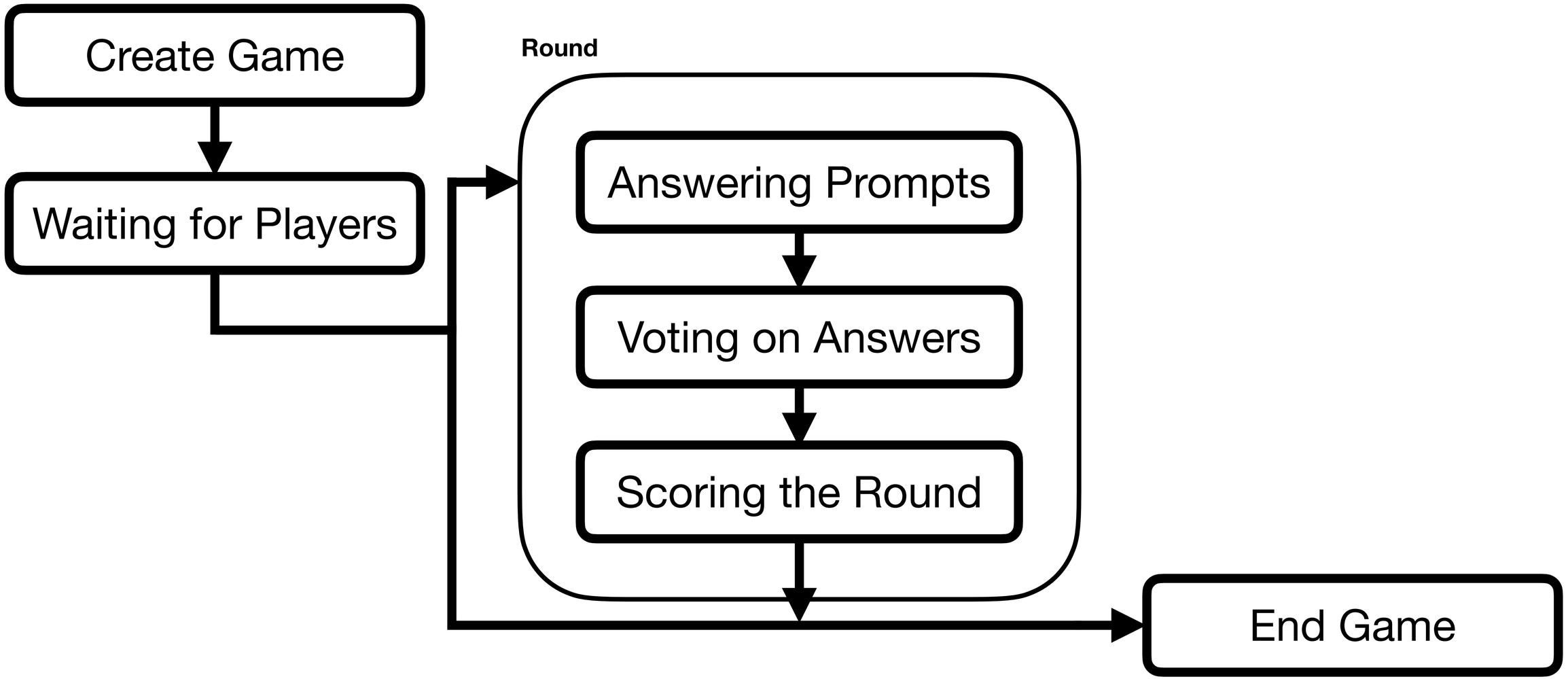
About today…


Sebastian
Do you have any Toyota Priuses?
less than a minute ago
Sebastian
We have 100+ Toyota Prius models available. To narrow down your options, do you have a specific price range in mind? Or perhaps a preferred color?
less than a minute ago
2010 Toyota Prius
l • 115k mi
$13,590
Ask Anything

Enter Quiplash

Enter ~~Quiplash~~ JestClout

The JestClout State Machine



This page intentionally left blank

Go Concurrency

~~Microthreads~~ Goroutines

```
package main

func doWork() {
  // Do work here.
}

func main() {

  go doWork()

}
```

Go Concurrency

Goroutines

```
package main

func main() {

  go func() {
    // Do work here.
  }()

}
```

Go Concurrency

Goroutines

```go
package main

func main() {

  nums := []int{1, 2, 3, 4, 5}
  for _, n := range nums {
    go func() {
      fmt.Println(n)
    }()
  }

}
```

```go
package main

func main() {

  nums := []int{1, 2, 3, 4, 5}
  for _, n := range nums {
    go func(num int) {
      fmt.Println(num)
    }(n)
  }

}
```

Go Concurrency

Channels

```
package main

import "fmt"

func doWork(in chan int) {
  for n := range in {
      // Do work here.
    }
  }
}
```

```
func main() {
  in := make(chan int)
  go doWork(in)

  in ← 1
  in ← 2
  in ← 3
}
```

Go Concurrency

Channels

```
package main

import "fmt"

func doWork(in chan int) {
  for n := range in {
      // Do work here.
    }
  }
}
```

```
func main() {
  in := make(chan int, 10)
  go doWork(in)

  in ← 1
  in ← 2
  in ← 3
}
```

Go Concurrency

Channels

```
package main

import "fmt"

func doWork(in chan int) {
  for {
    select {
    case n := <-in:
      // Do work here.
    }
  }
}
```

```
func main() {
  in := make(chan int)
  go doWork(in)

  in <- 1
  in <- 2
  in <- 3
}
```

Go Concurrency

Channels

```
package main

import "fmt"

type Command struct {
  n          int
  resultChan chan int
}

func double(in chan *Command) {
  for req := range in {
    req.resultChan ← req.n * 2
  }
}
```

```
func main() {
  in := make(chan *Command)
  go double(in)

  req := &Command{3, make(chan int)}

  in ← req

  fmt.Printf("answer: %d\n", ←req.resultChan)
}
```

Go Concurrency

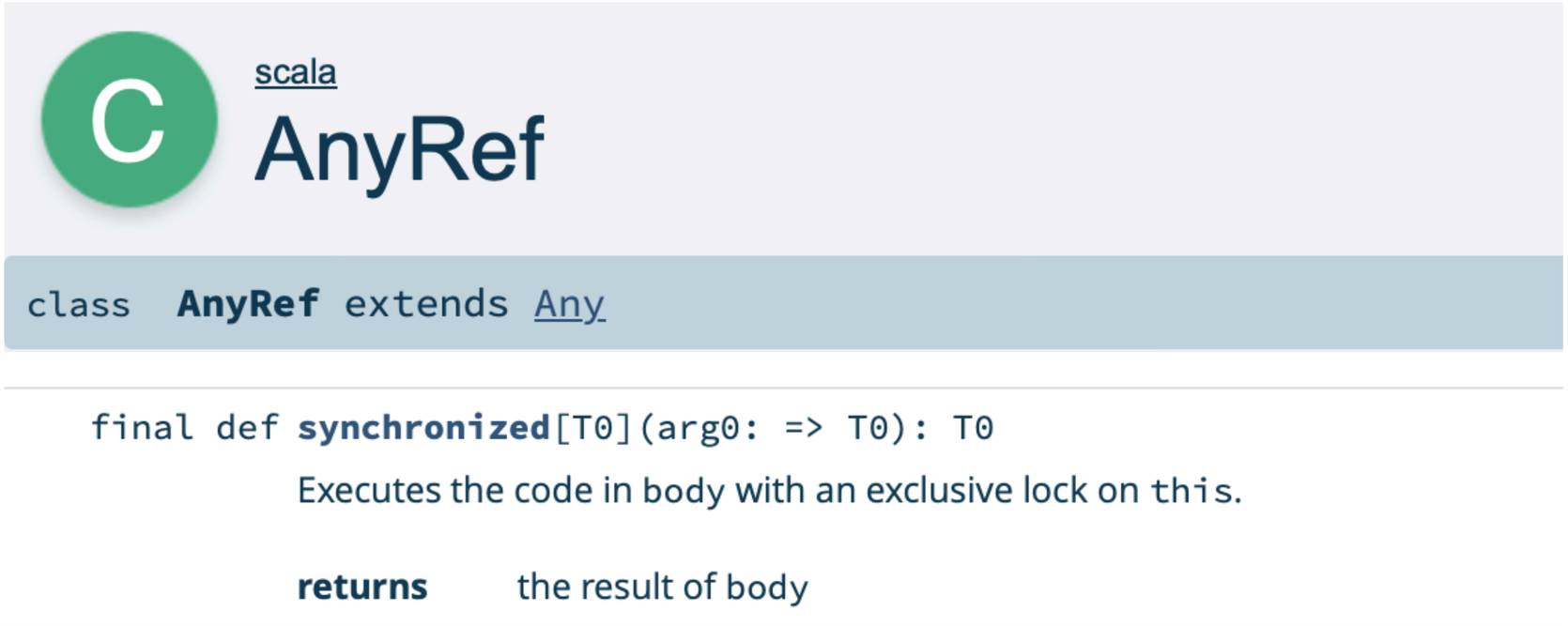
Mutexes

```
package main

import "sync"

type Datastore struct {
  Data map[string]string
  mu sync.Mutex
}

func (d *Datastore) doWork() {
  d.mu.Lock()
  defer d.mu.Unlock()

  // Do work here.
}
```

```
func main() {
  d := &Datastore{}
  go func() {
    d.doWork()
  }()
}
```

Scala Concurrency

Mutexes



scala

AnyRef

```
class  AnyRef extends Any
```

```
final def synchronized[T0](arg0: => T0): T0
```

Executes the code in body with an exclusive lock on `this`.

returns the result of body

Scala Concurrency

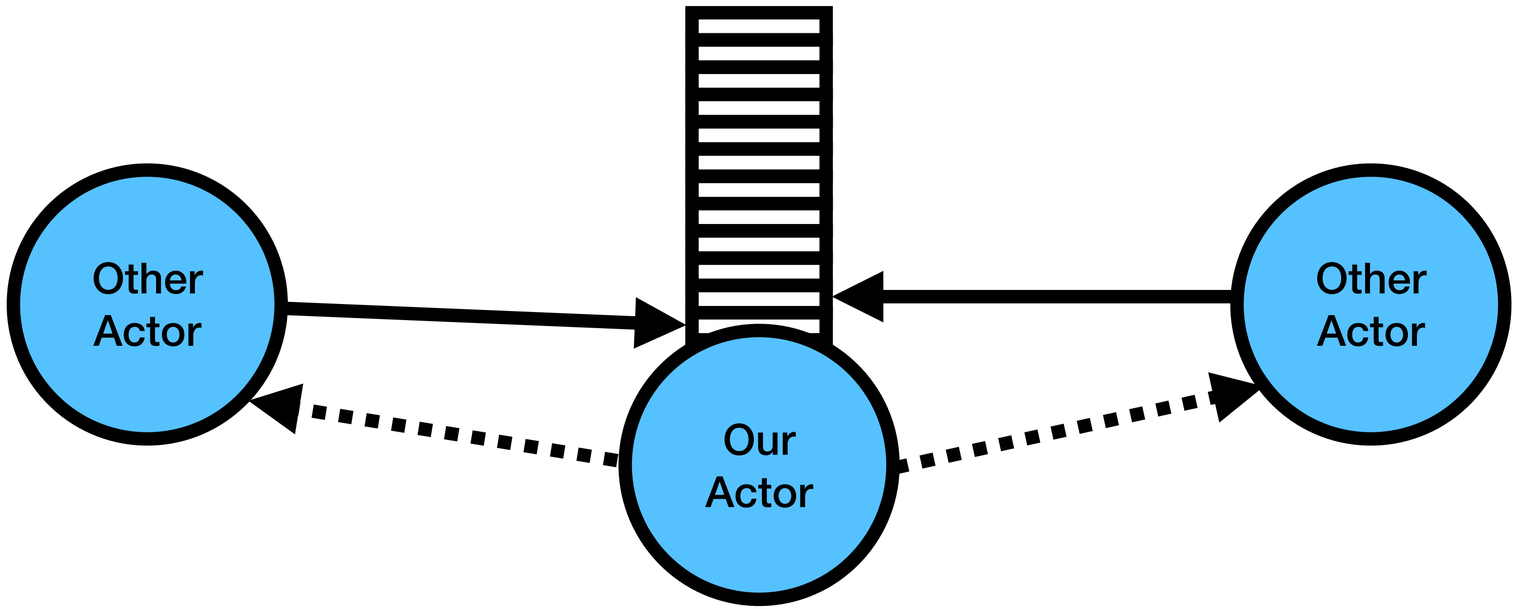
Microthreads Java Loom

Futures

```scala
import scala.concurrent._
import scala.concurrent.duration._
import scala.util.{Failure, Success}

def double(n: Int)(implicit ec: ExecutionContext): Future[Int] =
  Future {
    n * 2
  }

object main extends App {
  import scala.concurrent.ExecutionContext.Implicits.global

  val n = 10
  val f1 = double(n)
  val f2 = double(n)

  f1.onComplete {
    case Success(result) ⇒ println(s"f1 result: $result")
    case Failure(ex)     ⇒ println(ex)
  }

  val result = Await.result(f2, 5.milliseconds)
  println(s"f2 result: $result")
}
```

Scala Concurrency

Actors



https://pekko.apache.org/

Scala Concurrency

Actors

```
import org.apache.pekko.actor.typed.{ActorRef, ActorSystem, Behavior, Scheduler}
import org.apache.pekko.actor.typed.scaladsl.Behaviors
import org.apache.pekko.actor.typed.scaladsl.AskPattern._
import org.apache.pekko.util.Timeout

import scala.concurrent.duration._
```

```scala
object Counter {
  sealed trait Command
  final case class CurrentValue(replyTo: ActorRef[Int]) extends Command
  final case object Increment extends Command
  final case class IncrementBy(n: Int) extends Command

  def apply(): Behavior[Command] =
    receive(0)

  private def receive(sum: Int): Behavior[Command] = {
    Behaviors.receiveMessage {
      case CurrentValue(replyTo) ⇒
        replyTo ! sum
        Behaviors.same

      case Increment ⇒
        receive(sum + 1)

      case IncrementBy(n) ⇒
        receive(sum + n)
    }
  }
}
```

```scala
object main extends App {
  val counter: ActorSystem[Counter.Command] = ActorSystem(Counter(), "CounterActor")

  implicit val timeout: Timeout = Timeout(5.seconds)
  implicit val scheduler: Scheduler = counter.scheduler

  counter ! Counter.Increment
  counter ? Counter.CurrentValue

  counter ! Counter.IncrementBy(5)
  counter.ask(Counter.CurrentValue)
}
```

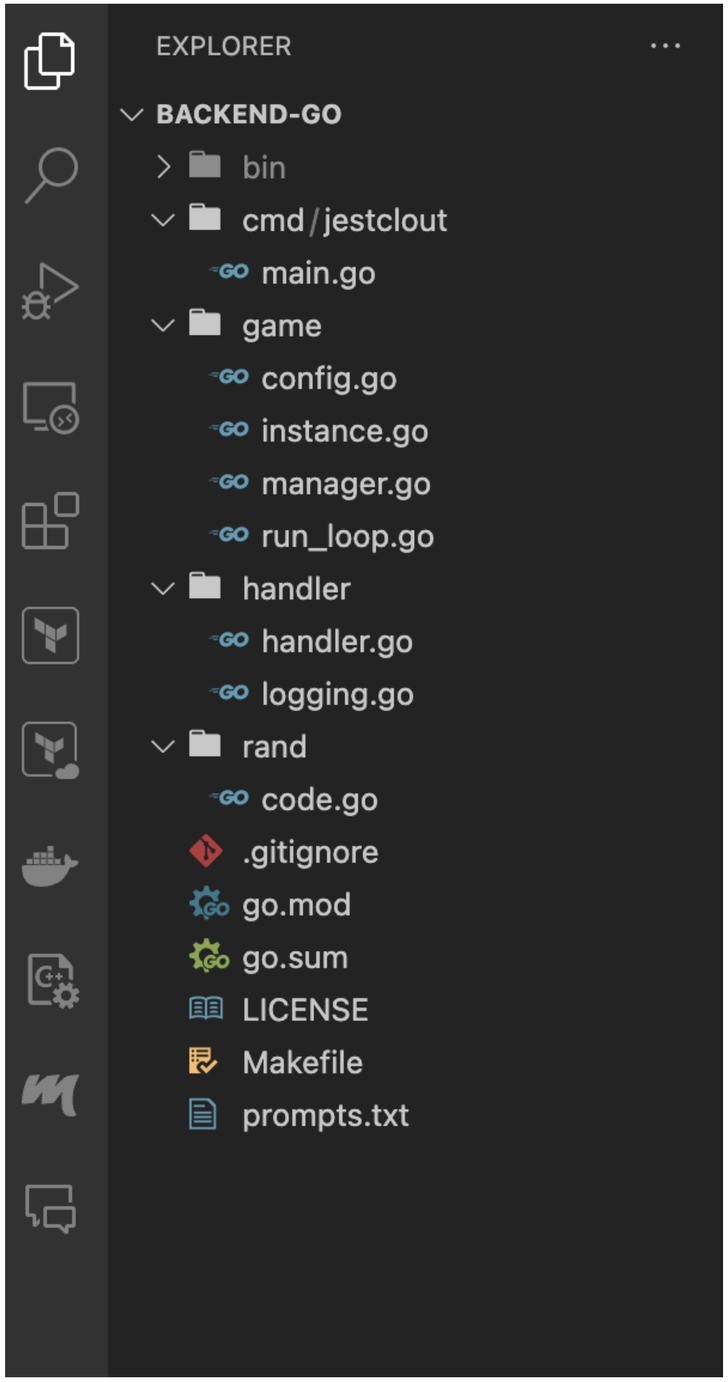
Scala Concurrency

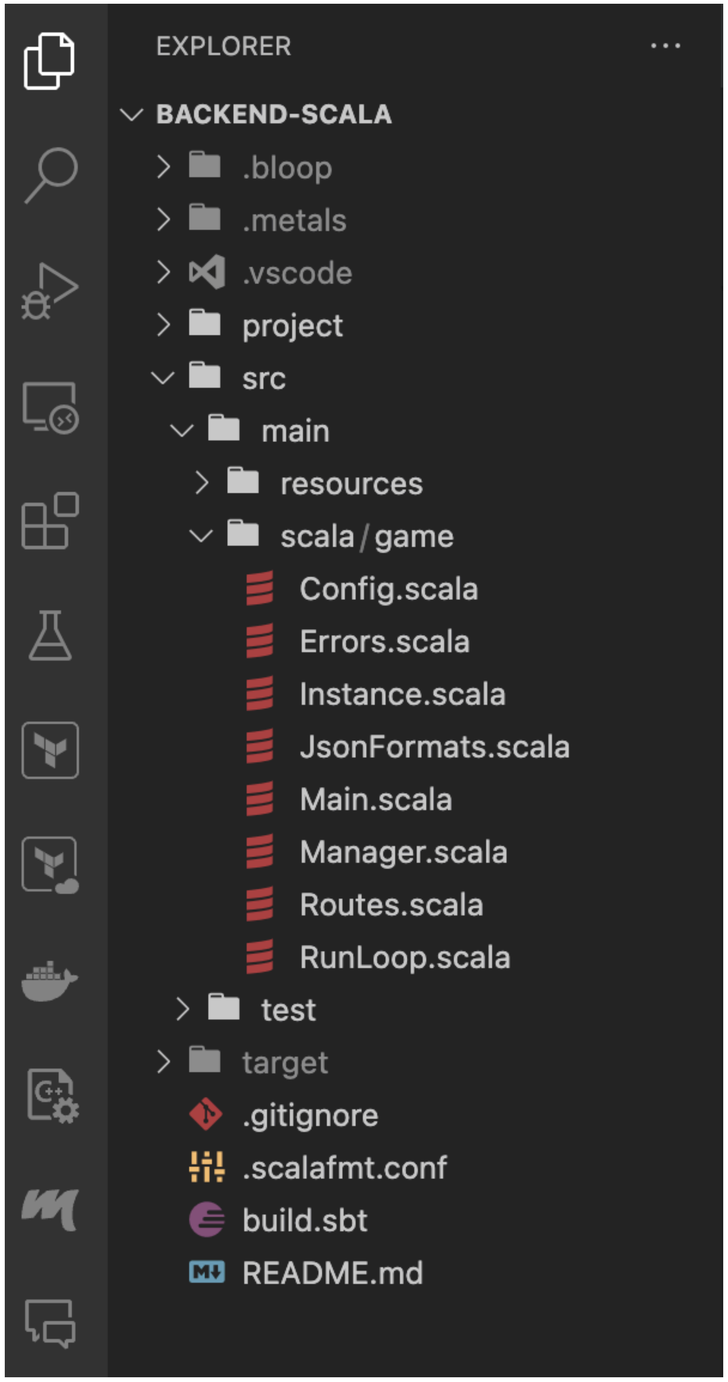
Actors from a template

```
sbt new apache/pekko-quickstart-scala.g8
```

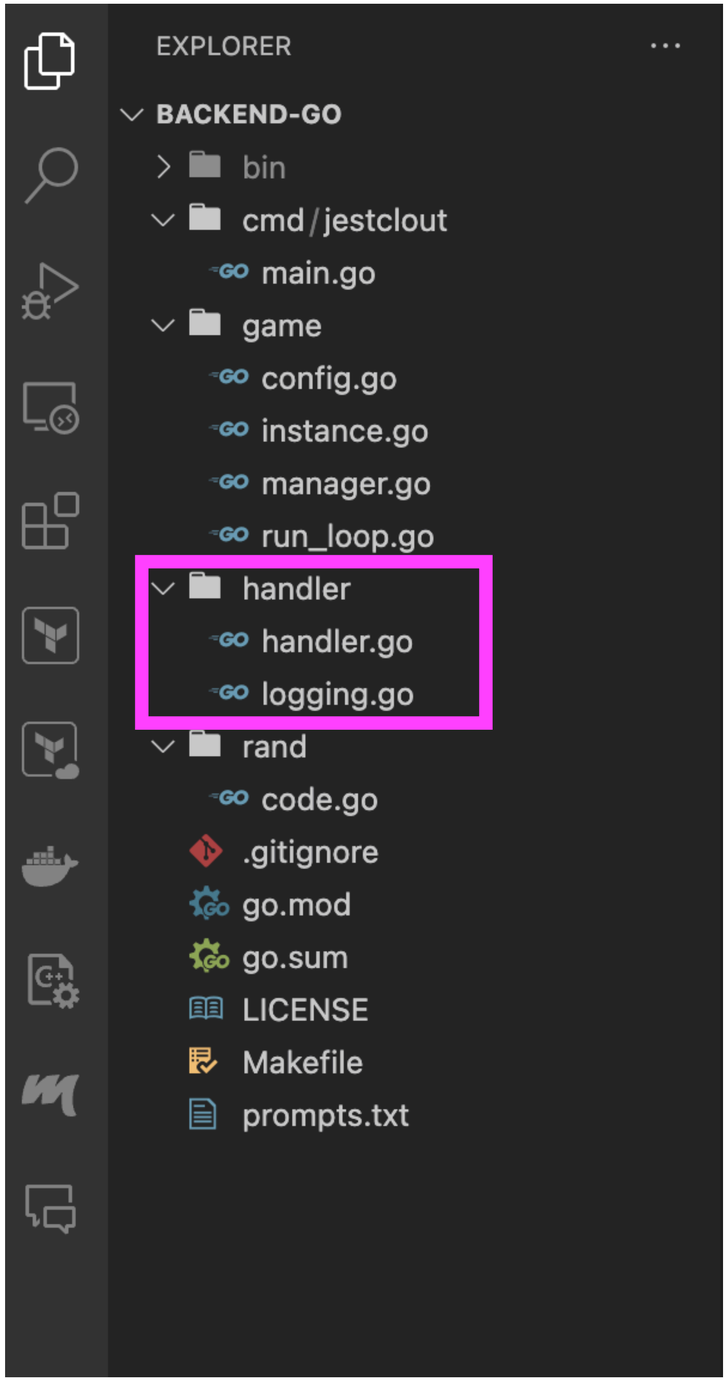
```
sbt new apache/pekko-http-quickstart-scala.g8
```

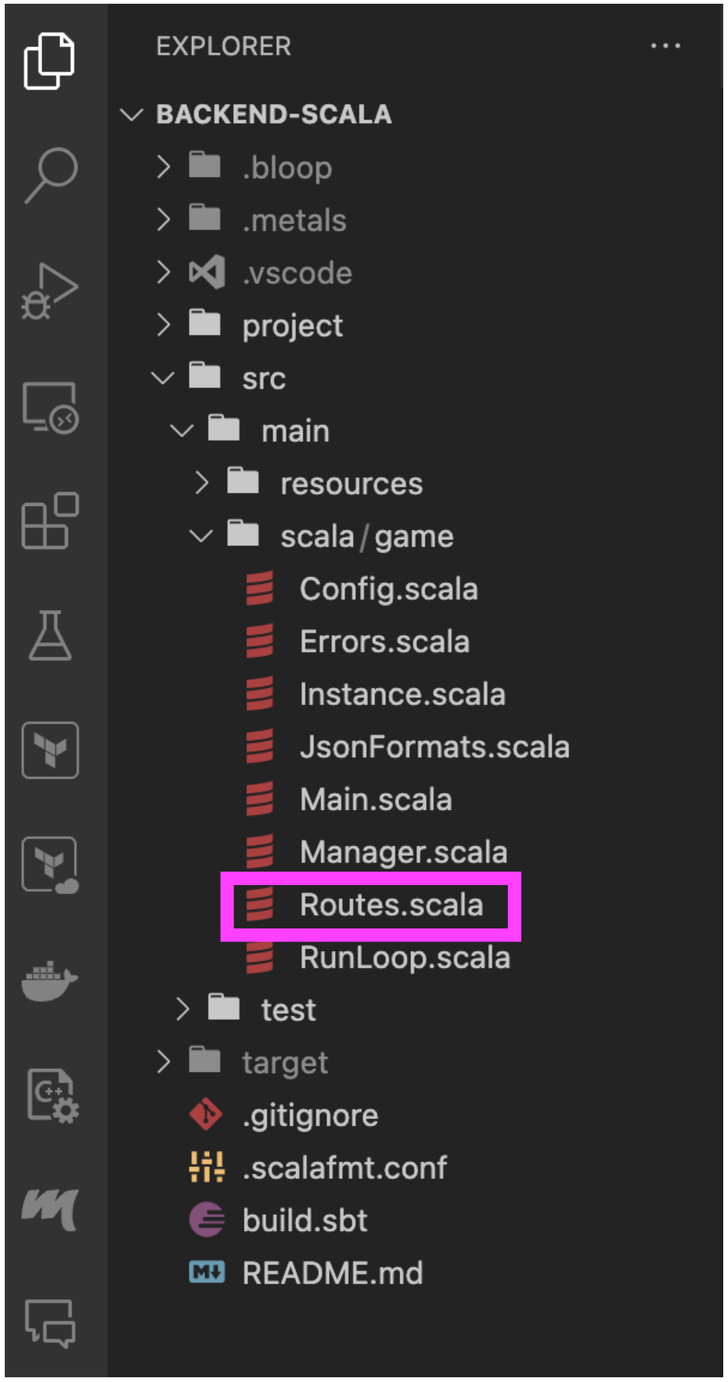
Structure


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt

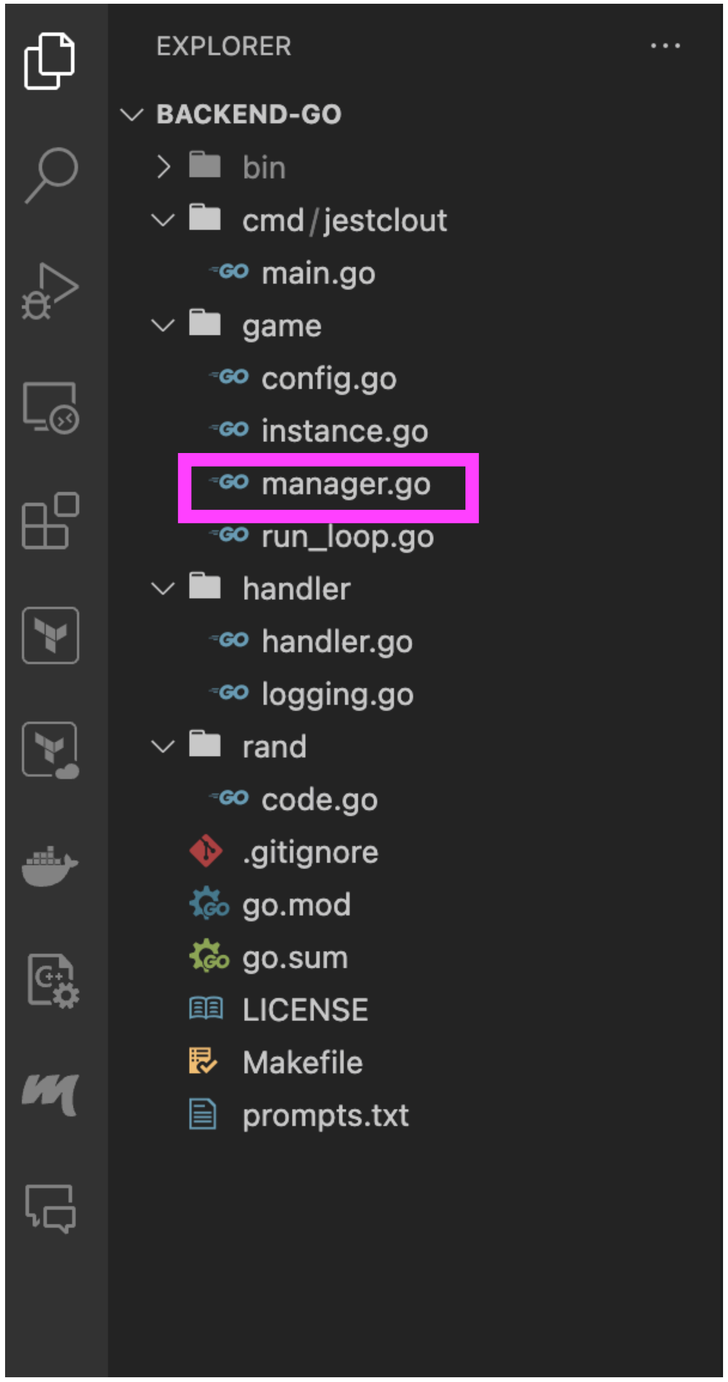

EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

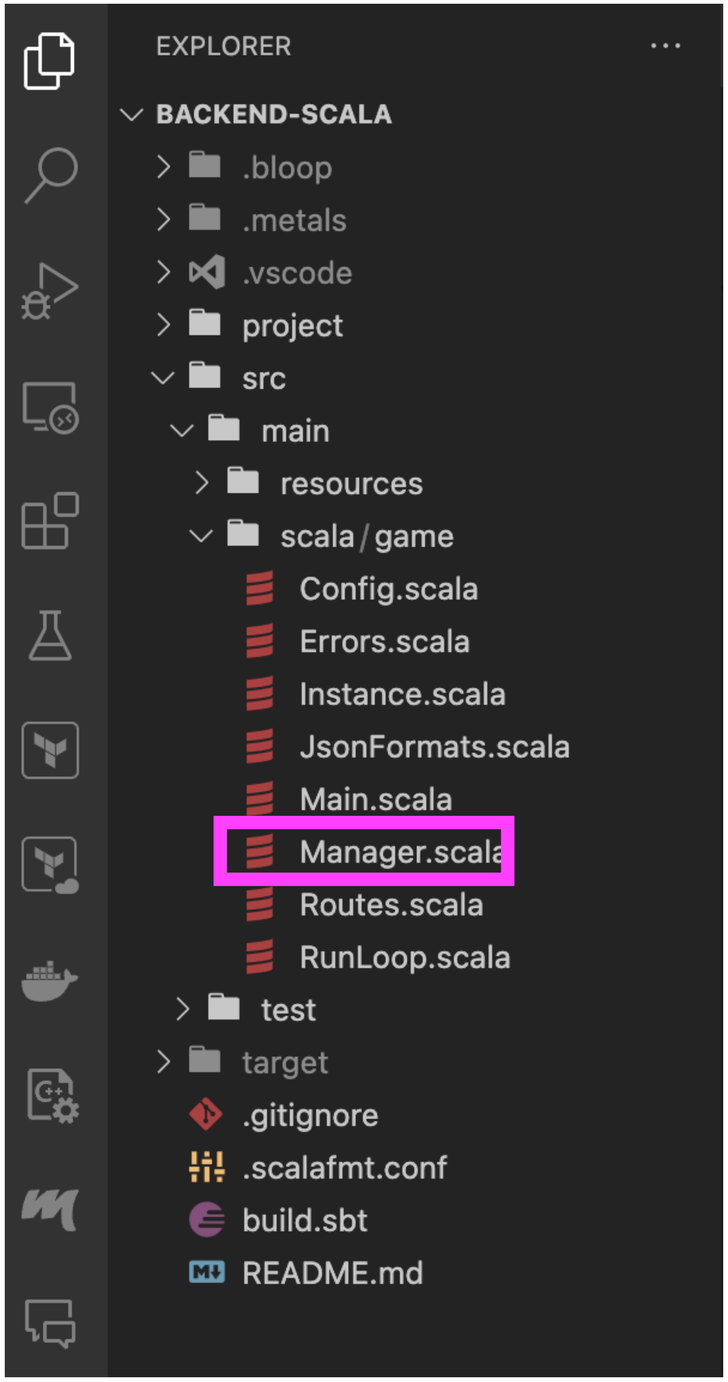
Structure


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt

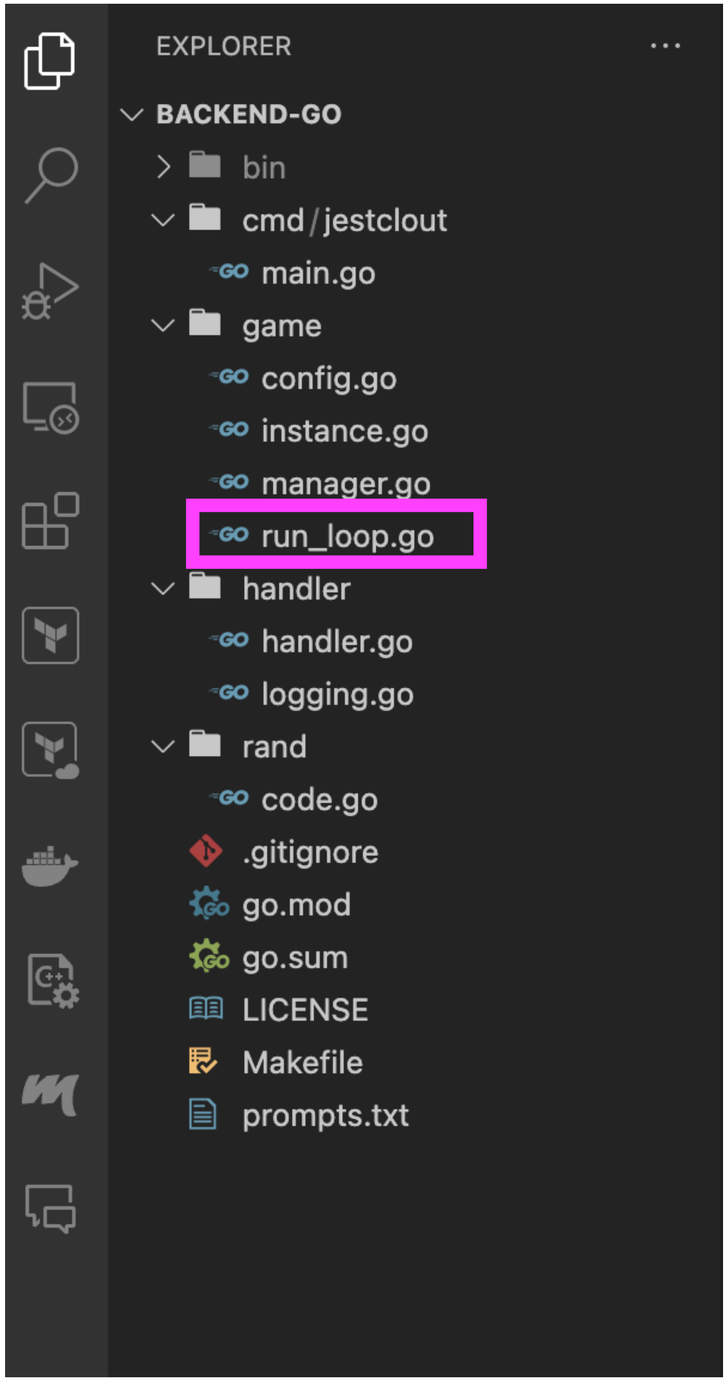

EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

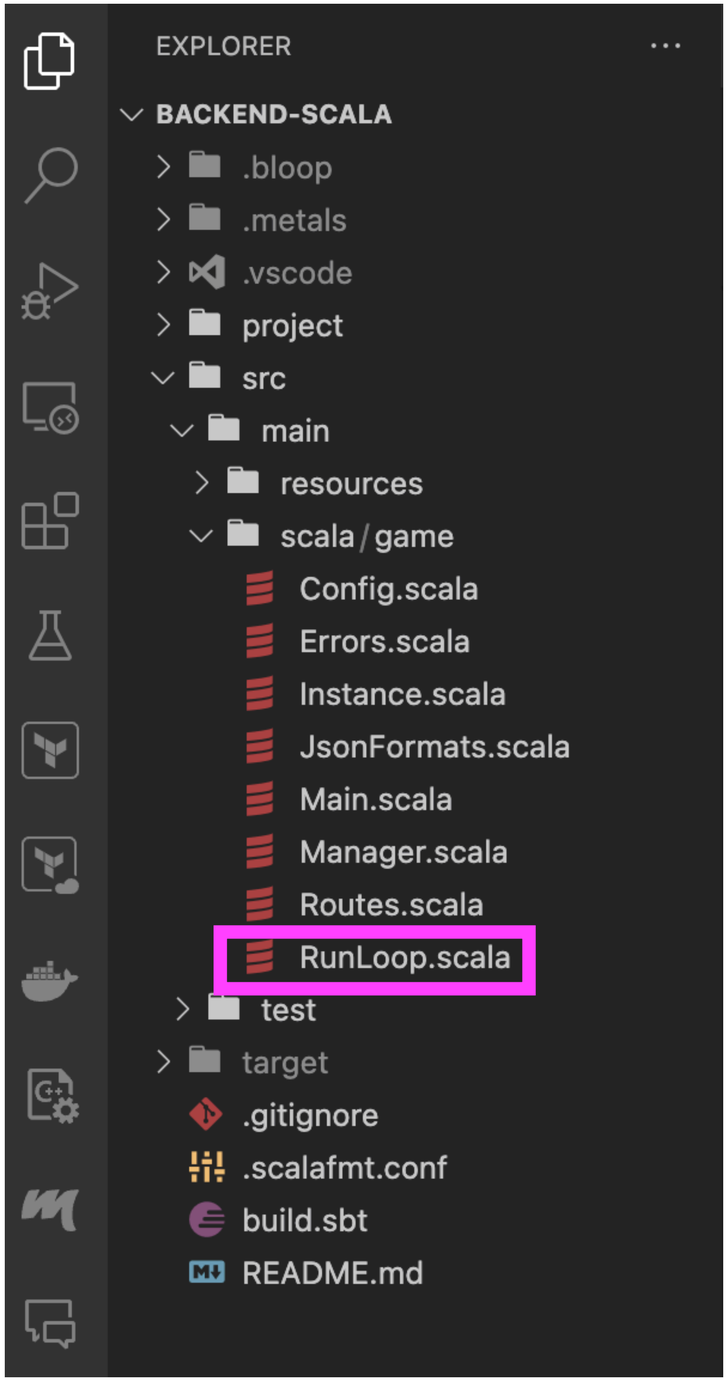
Structure


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt

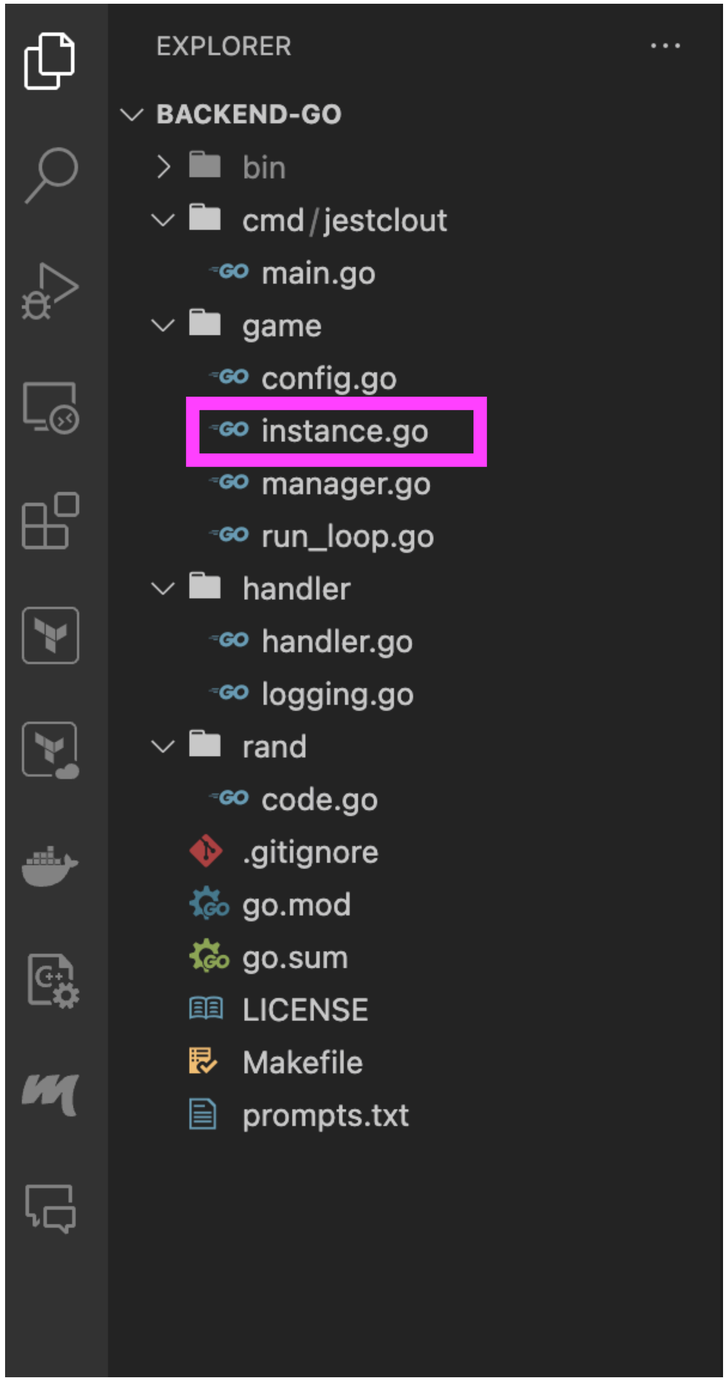

EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

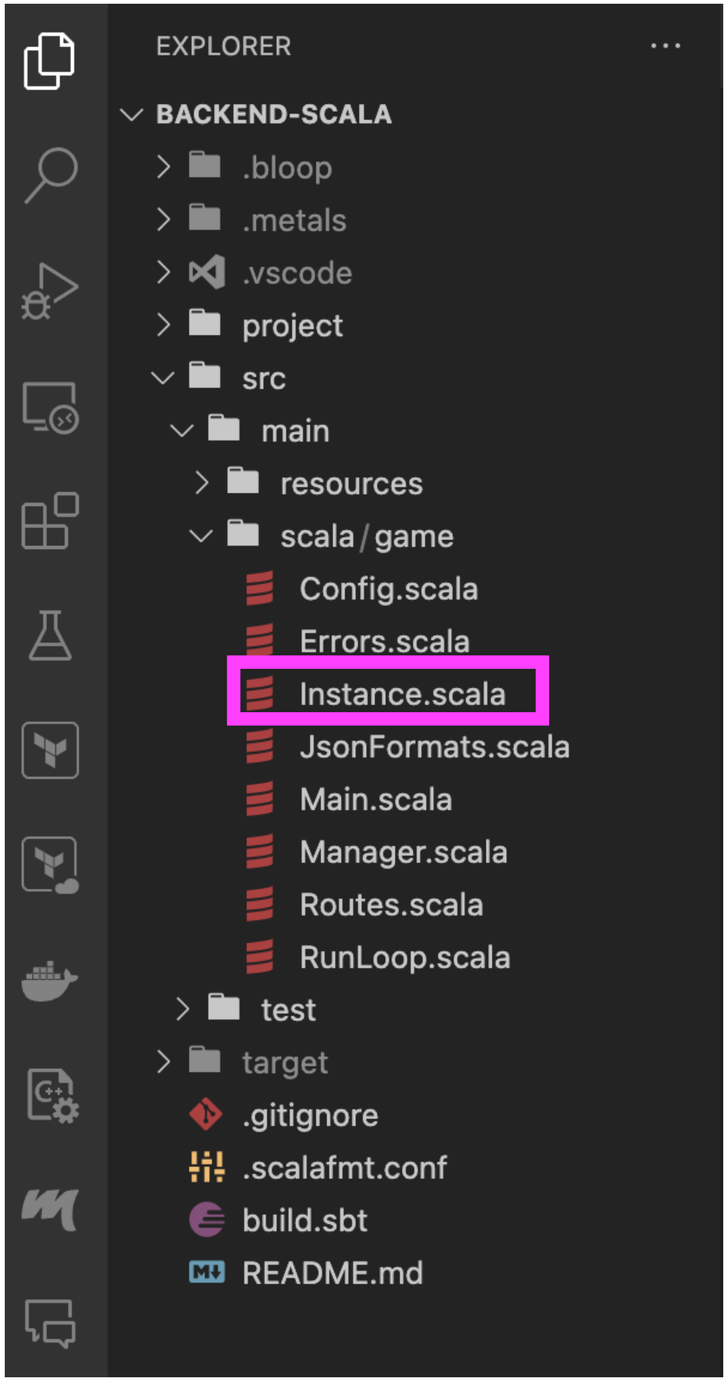
Structure


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt

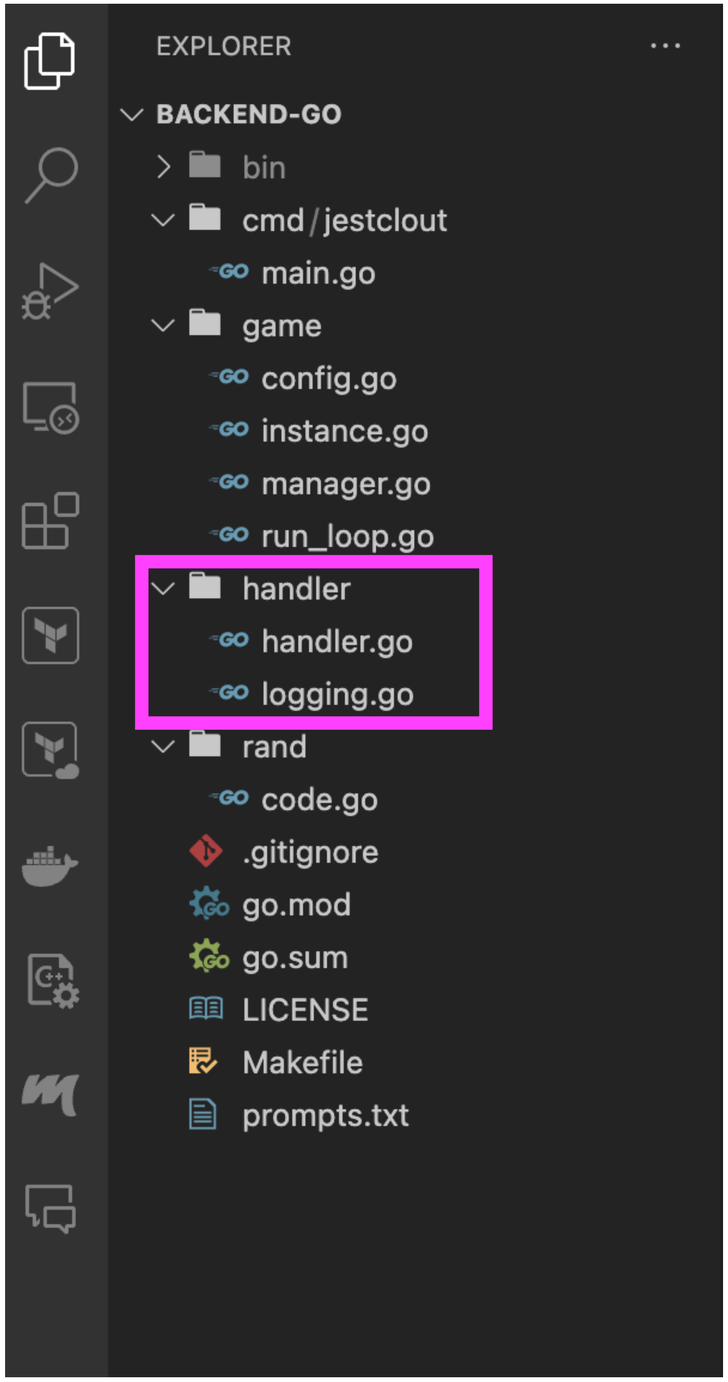

EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

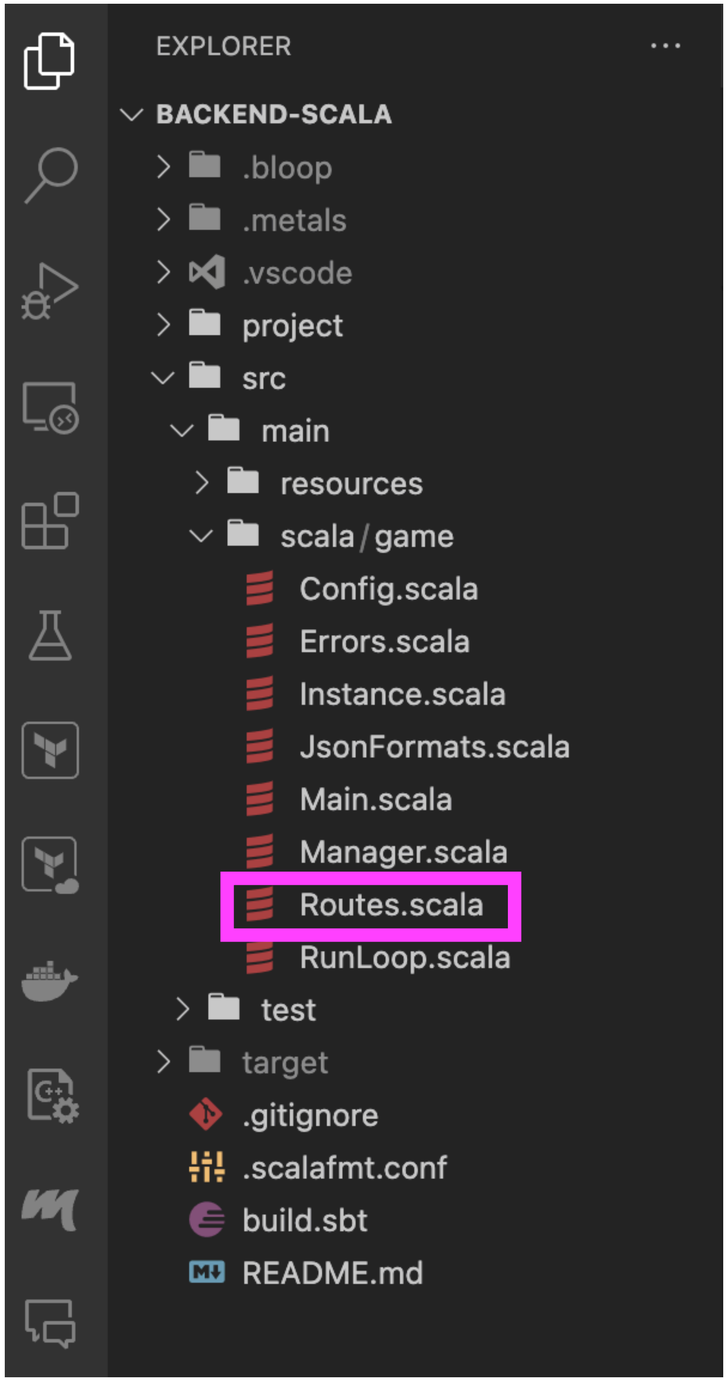
Structure


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt


EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

Implementation


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt


EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

```go
import "github.com/gorilla/mux"

type Handler struct {
  GameManager *game.Manager
  Logger      zerolog.Logger
}

//  ...

func (h *Handler) Router() *mux.Router {
  r := mux.NewRouter()

  api := r.PathPrefix("/api/v1").Subrouter()
  api.Use(h.LogRequest)

  api.HandleFunc("/game", h.CreateGame).Methods("POST")
  api.HandleFunc("/game/{gameCode}", h.GetGameState).Methods("GET")
  api.HandleFunc("/game/{gameCode}", h.ExecCommand).Methods("POST")

  return r
}
```

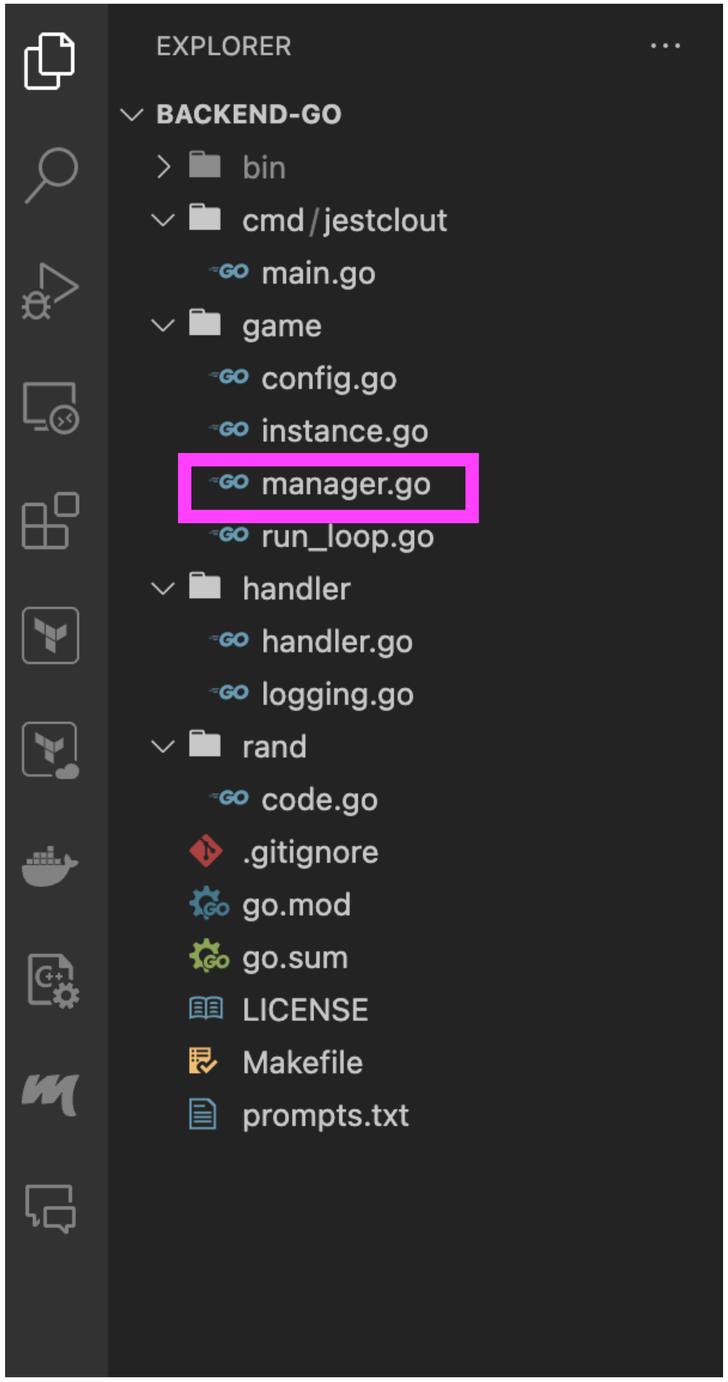
```go
func (h *Handler) CreateGame(w http.ResponseWriter, r *http.Request) {
  newGame, err := h.GameManager.NewGame()
  if err ≠ nil {
    http.Error(w, "", http.StatusInternalServerError)
    return
  }

  state := newGame.GetPublicState(0)

  payload, err := json.Marshal(state)
  if err ≠ nil {
    http.Error(w, "", http.StatusInternalServerError)
    return
  }

  w.WriteHeader(http.StatusCreated)
  w.Header().Set("Content-Type", "application/json")
  _, err = w.Write(payload)
  if err ≠ nil {
    // ...
  }
}
```

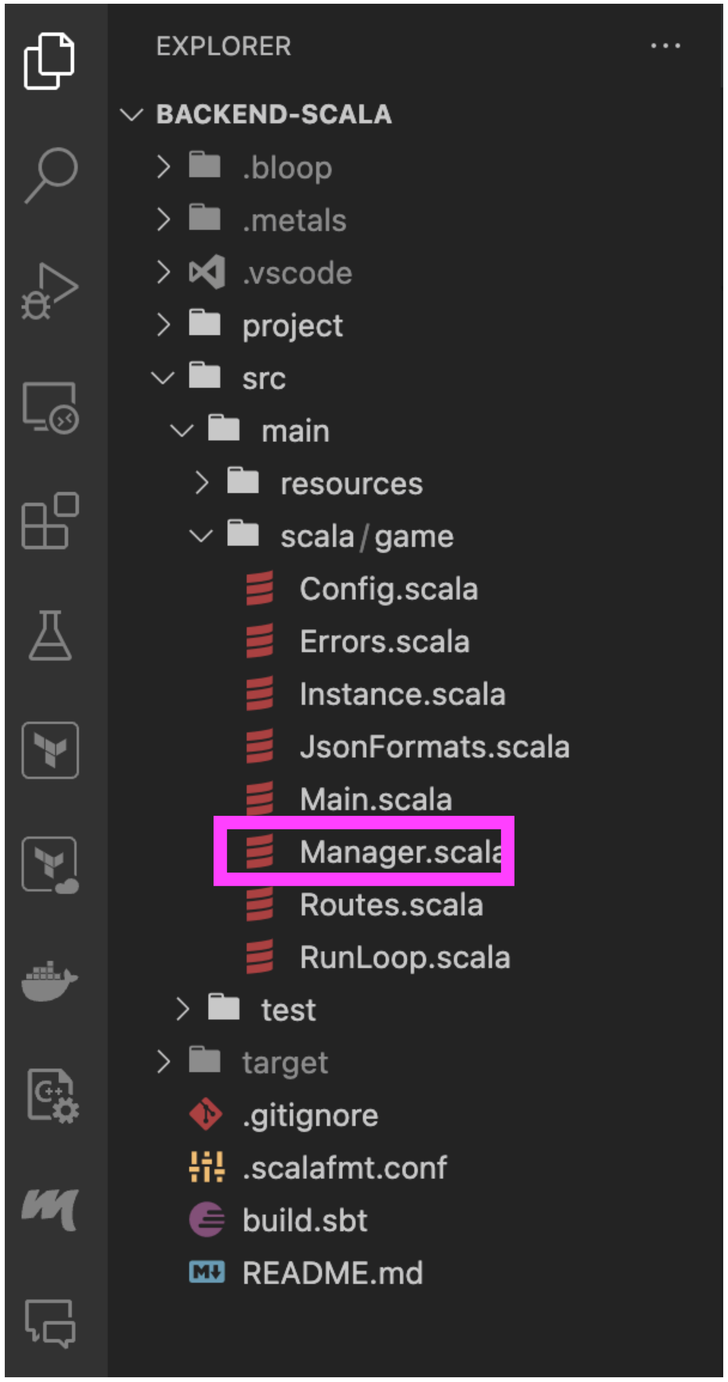
https://pekko.apache.org/docs/pekko-http/current/routing-dsl/index.html

```
def createGame(): Future[StatusReply[PublicGameState]] =
  gameManager.ask(Manager.CreateGame.apply)

val gameRoutes: Route =
  pathPrefix("api" / "v1" / "game") {
    concat(
      pathEnd {
        post {
          onSuccess(createGame()) { status ⇒
            status match {
              case StatusReply.Success(response: PublicGameState) ⇒
                complete((StatusCodes.Created, response))
              case _ ⇒
                complete(StatusCodes.InternalServerError)
            }
          }
        }
      }
    )
  }
```

```go
package handler

import (
  "encoding/json"
  "errors"
  "net/http"
  "strconv"

  "github.com/gorilla/mux"
  "github.com/rs/zerolog"
  "github.com/rs/zerolog/hlog"

  "github.com/jestclout/jestclout-go/game"
)

var (
  ErrCreateGame = errors.New("failed to create game")
)

type Handler struct {
  GameManager *game.Manager
  Logger      zerolog.Logger
}

func New(manager *game.Manager, ll zerolog.Logger) *Handler {
  return &Handler{
    GameManager: manager,
    Logger:      ll,
  }
}

func (h *Handler) PlayerIDFromRequest(r *http.Request) uint64 {
  playerHeader := r.Header.Get("X-Player-Id")
```

```scala
package game

import org.apache.pekko
import pekko.actor.typed.ActorRef
import pekko.actor.typed.ActorSystem
import pekko.actor.typed.scaladsl.AskPattern._
import pekko.http.scaladsl.model.StatusCodes
import pekko.http.scaladsl.server.Directive1
import pekko.http.scaladsl.server.Directives._
import pekko.http.scaladsl.server.Route
import pekko.pattern.StatusReply
import pekko.util.Timeout

import scala.concurrent.Future
import scala.util.Try

class JestCloutRoutes(gameManager: ActorRef[Manager.Command])(implicit
    val system: ActorSystem[_]
) {

  import pekko.http.scaladsl.marshallers.sprayjson.SprayJsonSupport._
  import JsonFormats._

  private implicit val timeout: Timeout = Timeout.create(
    system.settings.config.getDuration("jestclout.routes.ask-timeout")
  )

  def playerIdFromRequest: Directive1[Option[Long]] =
    optionalHeaderValueByName("X-Player-Id").map(_.flatMap(_.toLongOption))

  def createGame(): Future[StatusReply[PublicGameState]] =
    gameManager.ask(Manager.CreateGame.apply)

  def getState(
```


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt

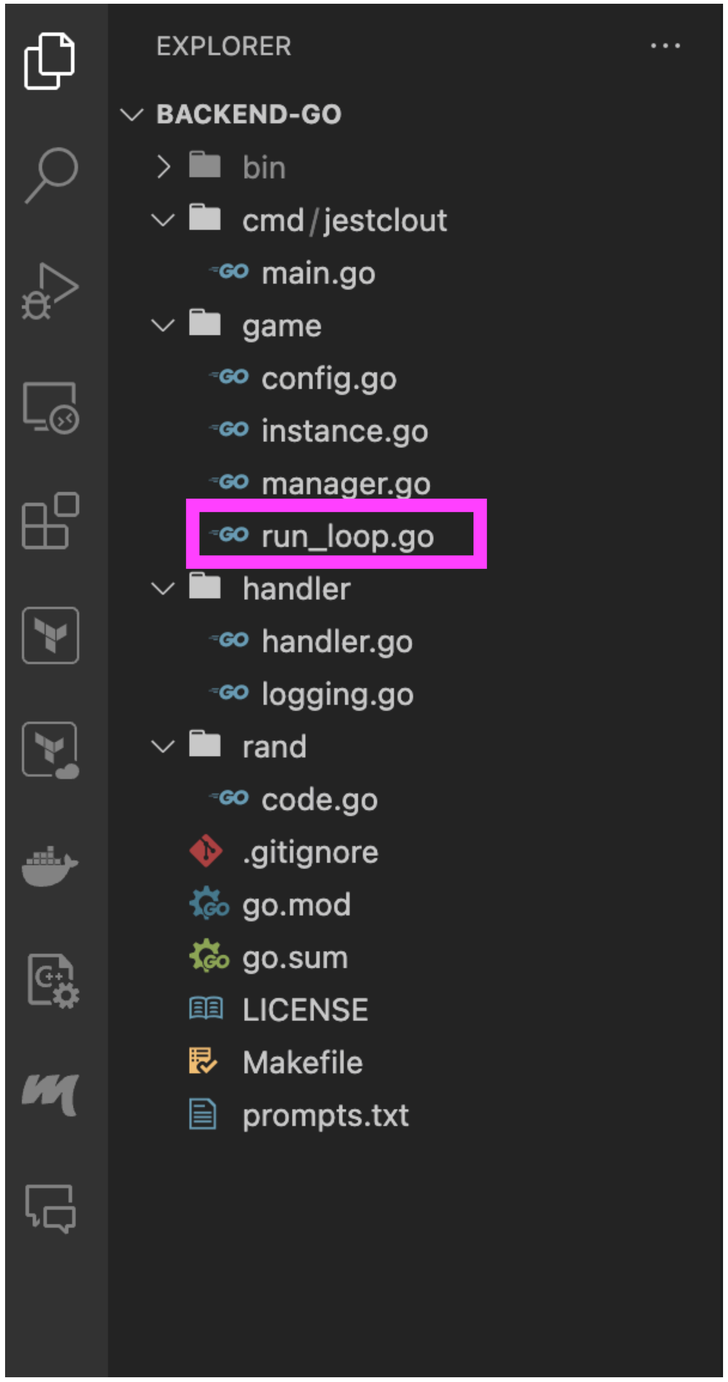

EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

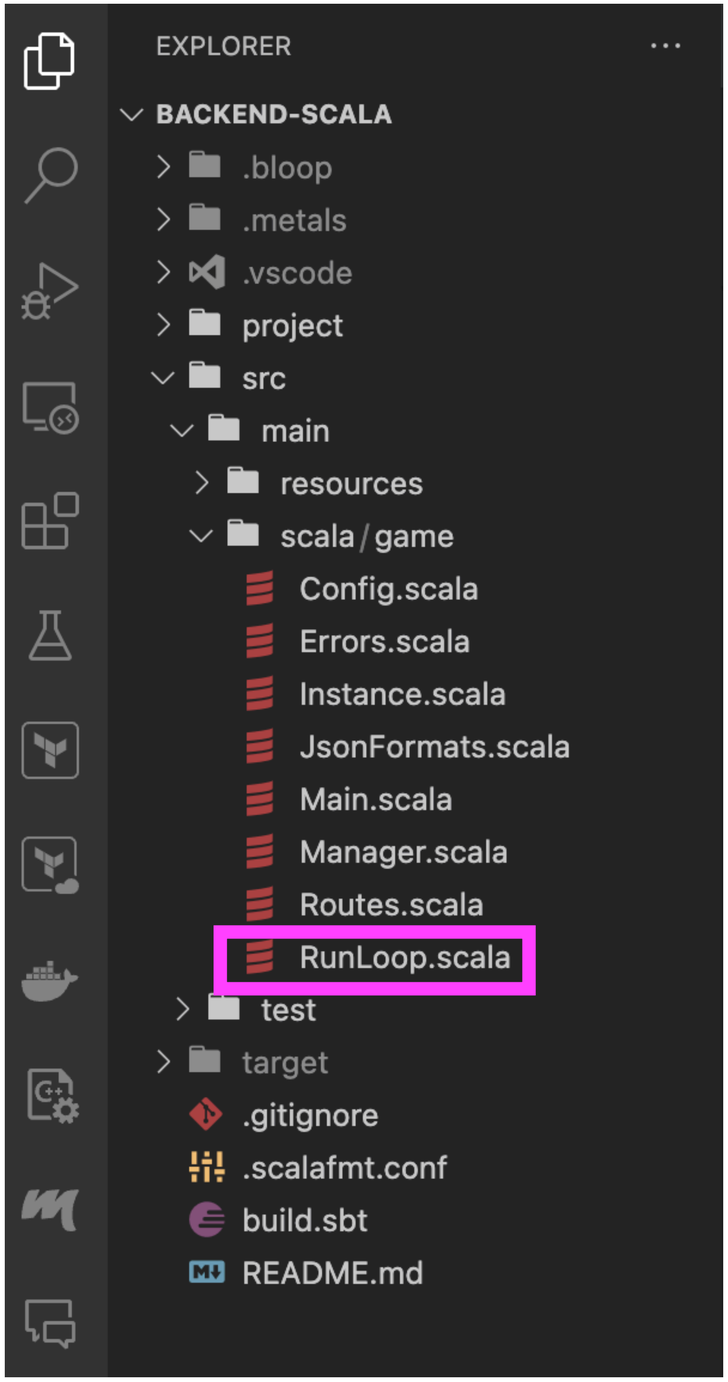
```
type Manager struct {
  // Has unexported fields.
}

func NewManager(prompts []string, config Config) *Manager
func (m *Manager) ExecCommand(code string, cmd Command) (*PublicGameState, error)
func (m *Manager) GetPublicGameState(code string, playerID uint64) (*PublicGameState, error)
func (m *Manager) NewGame() (*PublicGameState, error)
```

```go
func (m *Manager) getGame(code string) (*RunLoop, error) {
  m.mu.Lock()
  defer m.mu.Unlock()

  game, ok := m.games[code]
  if !ok {
    return nil, ErrGameNotFound
  }

  return game, nil
}

func (m *Manager) GetPublicGameState(code string, playerID uint64) (*PublicGameState, error)
{
  game, err := m.getGame(code)
  // ...
}

func (m *Manager) ExecCommand(code string, cmd Command) (*PublicGameState, error) {
  game, err := m.getGame(code)
  // ...
}
```

```
object Manager {

  sealed trait Command

  case class CreateGame(replyTo: ActorRef[StatusReply[PublicGameState]])
      extends Command

  case class GetPublicGameState(
      code: String,
      playerId: Option[Long],
      replyTo: ActorRef[StatusReply[PublicGameState]]
  ) extends Command

  case class ExecCommand(
      code: String,
      cmd: ManagerCmd,
      replyTo: ActorRef[StatusReply[PublicGameState]]
  ) extends Command

  // ...
}
```

```scala
def apply(prompts: List[String]): Behavior[Command] =
  Behaviors.setup { context ⇒
    manager(context, Map.empty, prompts)
  }

private def manager(
    context: ActorContext[Command],
    games: Map[String, ActorRef[RunLoop.Command]],
    prompts: List[String]
): Behavior[Command] = {

  // ...

  Behaviors.receiveMessage[Command] {
    case CreateGame(replyTo) ⇒
      // ...

      val runLoop = context.spawn(RunLoop(code, prompts), code)
      runLoop ! RunLoop.GetState(None, replyTo)

      val newGames = games + (code → runLoop)
      manager(context, newGames, prompts)

  // ...
```


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt


EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

```go
type CommandType int

const (
  GetState CommandType = iota
  AddPlayer
  UpdatePlayer
  RemovePlayer
  StartGame
  AnswerPrompt
  // ...
)

type Command struct {
  Type     CommandType `json:"cmdType"`
  PlayerID uint64      `json:"playerId"`
  Player   *Player     `json:"player"`
  PromptID uint64      `json:"promptId"`
  Answer   *Answer     `json:"answer"`
  // ...
}
```

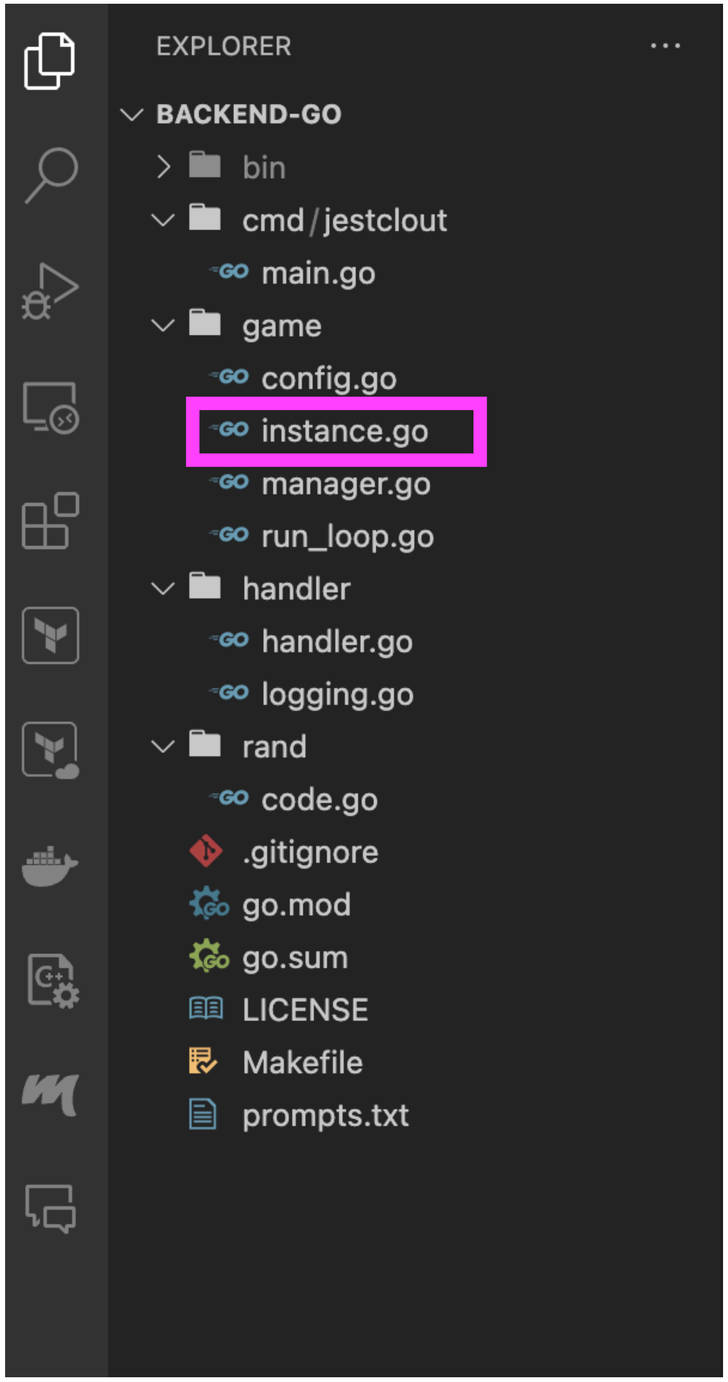
```scala
object Commands extends Enumeration {
  type CommandType = Value

  val GetState = Value
  val AddPlayer = Value
  val UpdatePlayer = Value
  val RemovePlayer = Value
  val StartGame = Value
  val AnswerPrompt = Value
  // ...
}

case class ManagerCmd(
    cmdType: Commands.Cmd,
    playerId: Option[Long] = None,
    player: Option[Player] = None,
    promptId: Option[Long] = None,
    answer: Option[Answer] = None
    // ...
)
```

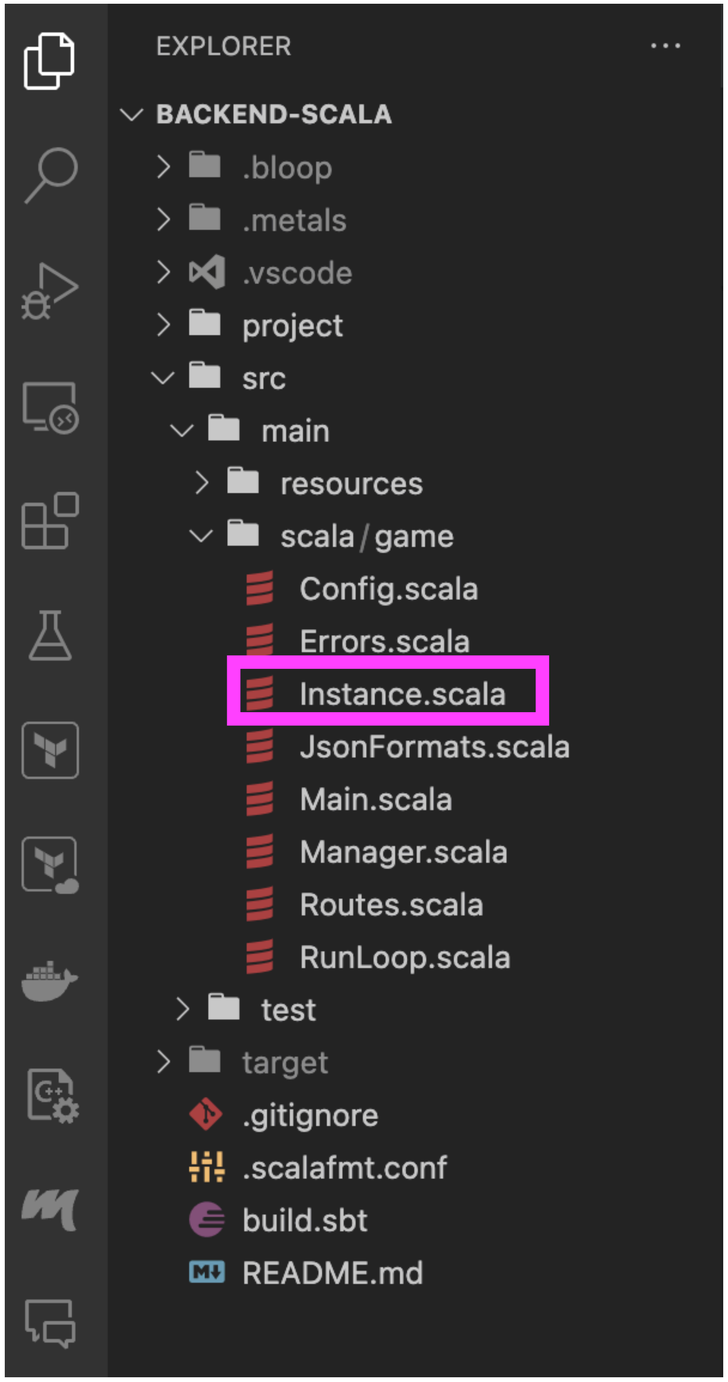
```go
func (g *RunLoop) ExecCommand(cmd Command) (*PublicGameState, error) {
  g.mu.Lock()
  defer g.mu.Unlock()

  instance := g.Instance

  switch cmd.Type {
  case AddPlayer:
    if cmd.Player == nil {
      return nil, ErrCmdMissingPlayer
    }

    err := instance.AddPlayer(cmd.Player)
    if err ≠ nil {
      return nil, err
    }
  // ...
  }

  return instance.GetState(cmd.PlayerID), nil
}
```

```
case class ManagerCmd(
    cmdType: Commands.Cmd,
    playerId: Option[Long] = None,
    player: Option[Player] = None,
    promptId: Option[Long] = None,
    answer: Option[Answer] = None
    // ...
) {

  def asRunLoopCmd(
      replyTo: ActorRef[StatusReply[PublicGameState]]
  ): RunLoop.Command =
    cmdType match {
      case Commands.GetState ⇒
        RunLoop.GetState(playerId, replyTo)

      case Commands.AddPlayer ⇒
        player match {
          case Some(p) ⇒ RunLoop.AddPlayer(p, replyTo)
          case _       ⇒ throw new PlayerNotFoundException()
        }
        // ...
```

```
case ExecCommand(code, cmd, replyTo) ⇒
  games.get(code) match {
    case Some(game) ⇒
      Try(cmd.asRunLoopCmd(replyTo)) match {
        case Success(runLoopCmd) ⇒
          game ! runLoopCmd
        case Failure(e) ⇒
          replyTo ! StatusReply.Error(e.getMessage)
      }

    case _ ⇒
      StatusReply.Error("game not found")
  }

  Behaviors.same
```

```
def apply(code: String, prompts: List[String]): Behavior[Command] =
  Behaviors.setup { context ⇒
    playersRunLoop(context, Instance(code, prompts))
  }

private def playersRunLoop(
    context: ActorContext[Command],
    instance: Instance
): Behavior[Command] =
  Behaviors.receiveMessage {
    case AddPlayer(player, replyTo) ⇒
      Try(instance.addPlayer(player)) match {
        case Success(newInstance) ⇒
          val publicState = newInstance.getState(player.id)
          replyTo ! StatusReply.Success(publicState)

          playersRunLoop(context, newInstance)

        case Failure(e) ⇒
          replyTo ! StatusReply.Error(e.getMessage)

          Behaviors.same
      }
    // ...
```

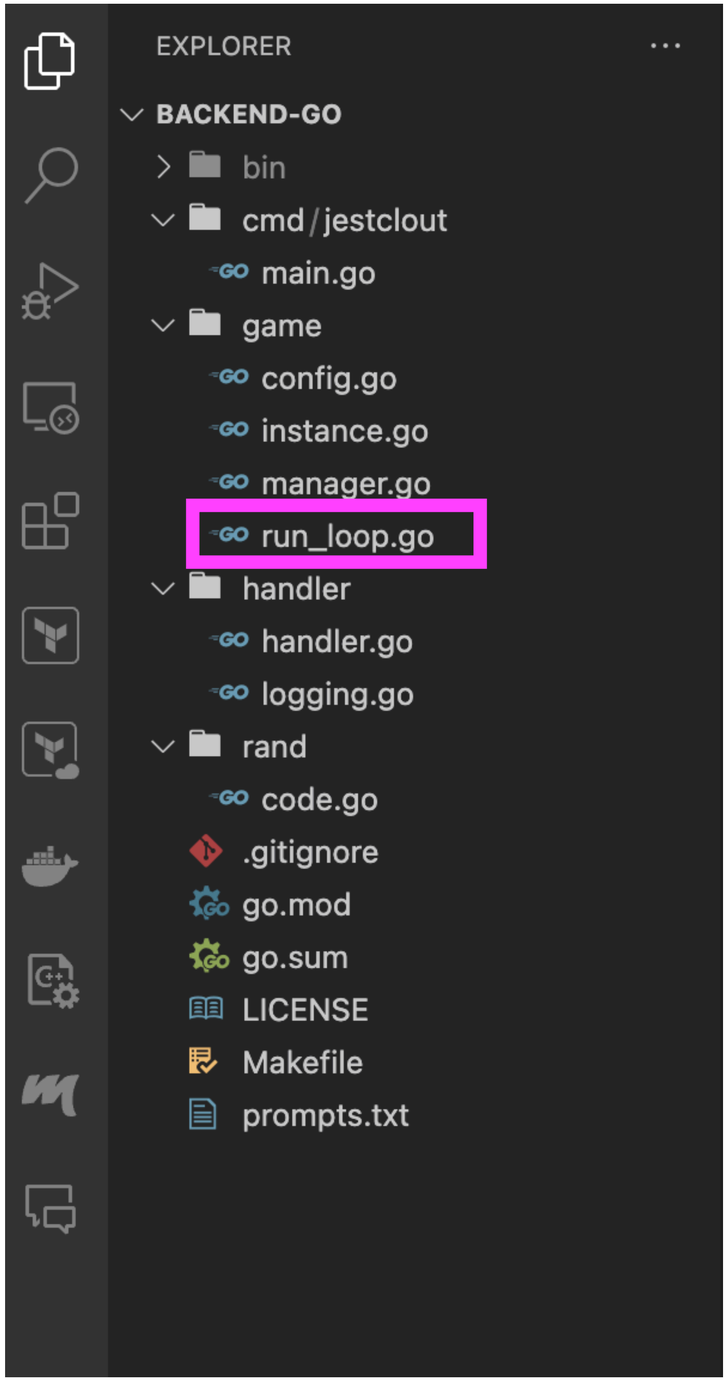

EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt

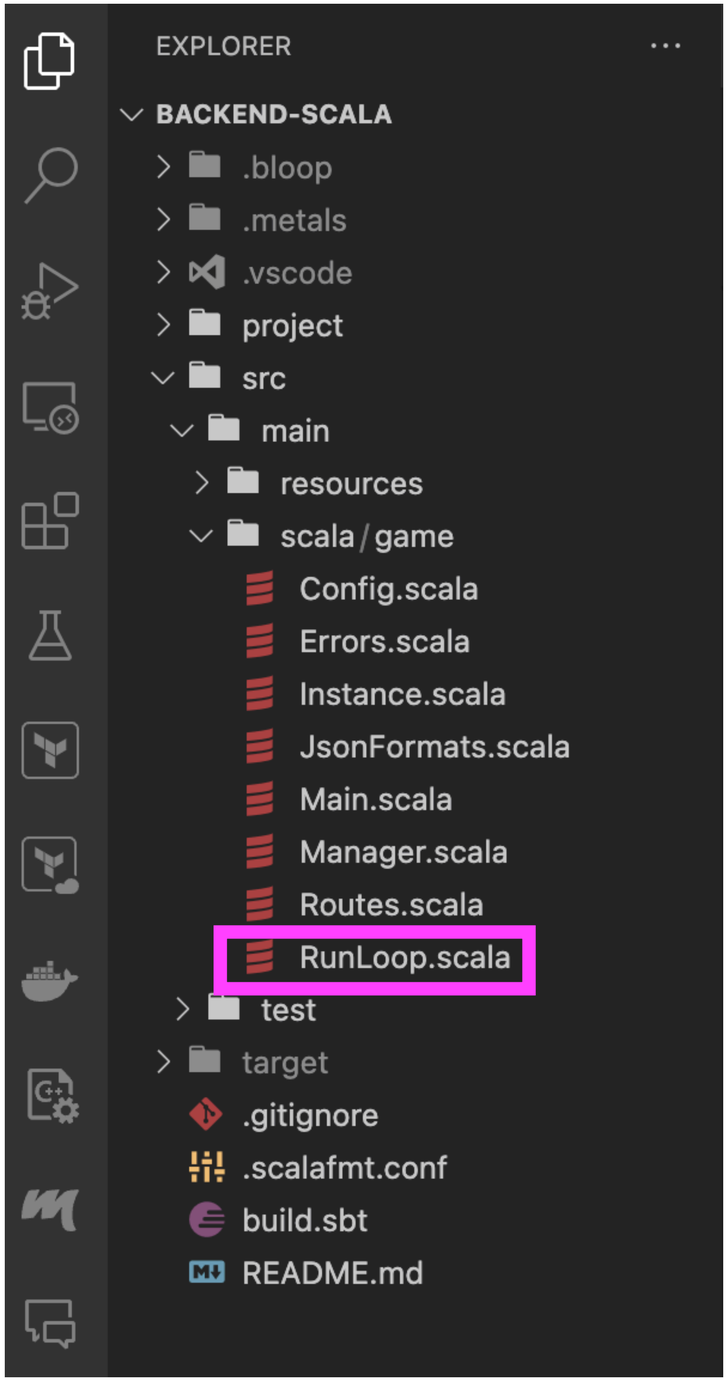

EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

```go
func (i *Instance) getNextUserID() uint64 {
  id := i.nextPlayerID
  i.nextPlayerID++

  return id
}

func (i *Instance) AddPlayer(p *Player) error {
  if len(i.players) ≥ i.config.MaxPlayers {
    return ErrPlayerLimitReached
  }

  if i.currentState ≠ WaitingForPlayers {
    return ErrGameInProgress
  }

  p.ID = i.getNextUserID()

  i.players = append(i.players, p)

  return nil
}
```

```
case class Instance(
  // ...
) {

  def addPlayer(player: Player): Instance = {
    if (players.length ⩾ config.maxPlayers) {
      throw new PlayerLimitReachedException()
    }

    if (currentState ≠ GameStates.WaitingForPlayers) {
      throw new GameInProgressException()
    }

    val newPlayer = Player(
      id = Some(nextPlayerID),
      name = player.name,
      score = Some(0)
    )

    copy(players = players :+ newPlayer, nextPlayerID = nextPlayerID + 1)
  }
```

```
  Try(instance.addPlayer(player)) match {
    case Success(newInstance) ⇒
      val publicState = newInstance.getState(player.id)
      replyTo ! StatusReply.Success(publicState)

      playersRunLoop(context, newInstance)

    case Failure(e) ⇒
      replyTo ! StatusReply.Error(e.getMessage)

      Behaviors.same
  }
// ...
```


EXPLORER
BACKEND-GO
bin
cmd / jestclout
main.go
game
config.go
instance.go
manager.go
run_loop.go
handler
handler.go
logging.go
rand
code.go
.gitignore
go.mod
go.sum
LICENSE
Makefile
prompts.txt


EXPLORER
BACKEND-SCALA
.bloop
.metals
.vscode
project
src
main
resources
scala / game
Config.scala
Errors.scala
Instance.scala
JsonFormats.scala
Main.scala
Manager.scala
Routes.scala
RunLoop.scala
test
target
.gitignore
.scalafmt.conf
build.sbt
README.md

```go
func (g *RunLoop) ExecCommand(cmd Command) (*PublicGameState, error) {
  g.mu.Lock()
  defer g.mu.Unlock()

  instance := g.Instance

  switch cmd.Type {
  // ...
  case StartGame:
    err := instance.AdvanceState()
    if err ≠ nil {
      return nil, err
    }

    // Set timeout for answering prompts.
    go func() {
      time.Sleep(120 * time.Second)
      g.AdvanceState(AnsweringPrompts, 0)
    }()
  // ...
```

```
func (g *RunLoop) AdvanceState(currentState State, currentVotingPrompt int) {
  g.mu.Lock()
  defer g.mu.Unlock()

  instance := g.Instance

  if instance.currentState == currentState {
    switch currentState {
    case VotingOnAnswers:
      if instance.currentVotingPrompt == currentVotingPrompt {
        instance.AdvanceState()
      }
    default:
      instance.AdvanceState()
    }
  }
}
```

```
object RunLoop {

  final case object AnsweringPromptsTimeout extends Command
  final case object VotingOnAnswersTimeout extends Command
  final case object ScoringRoundTimeout extends Command
  final case object TimeoutFailed extends Command

  // ...
}
```

```
private def playersRunLoop(
    context: ActorContext[Command],
    instance: Instance
): Behavior[Command] =
  Behaviors.receiveMessage {
    // ...

    case StartGame(replyTo) ⇒
      val newInstance = instance.advanceState
      val publicState = newInstance.getState()
      replyTo ! StatusReply.Success(publicState)

      implicit val ec: ExecutionContext = context.executionContext

      val timeout = Future(Thread.sleep(120.seconds.toMillis))
      context.pipeToSelf(timeout) {
        case Success(_) ⇒ AnsweringPromptsTimeout
        case Failure(_) ⇒ TimeoutFailed
      }

      answeringRunLoop(context, newInstance)

    case _ ⇒
      Behaviors.same
  }
```

```
def apply(code: String, prompts: List[String]): Behavior[Command] =
  Behaviors.setup { context ⇒
    playersRunLoop(context, Instance(code, prompts))
  }

private def playersRunLoop(
    context: ActorContext[Command],
    instance: Instance
): Behavior[Command] = ???

private def answeringRunLoop(
    context: ActorContext[Command],
    instance: Instance
): Behavior[Command] = ???

private def votingRunLoop(
    context: ActorContext[Command],
    instance: Instance
): Behavior[Command] = ???

// ...
```

Go vs Scala

Summary

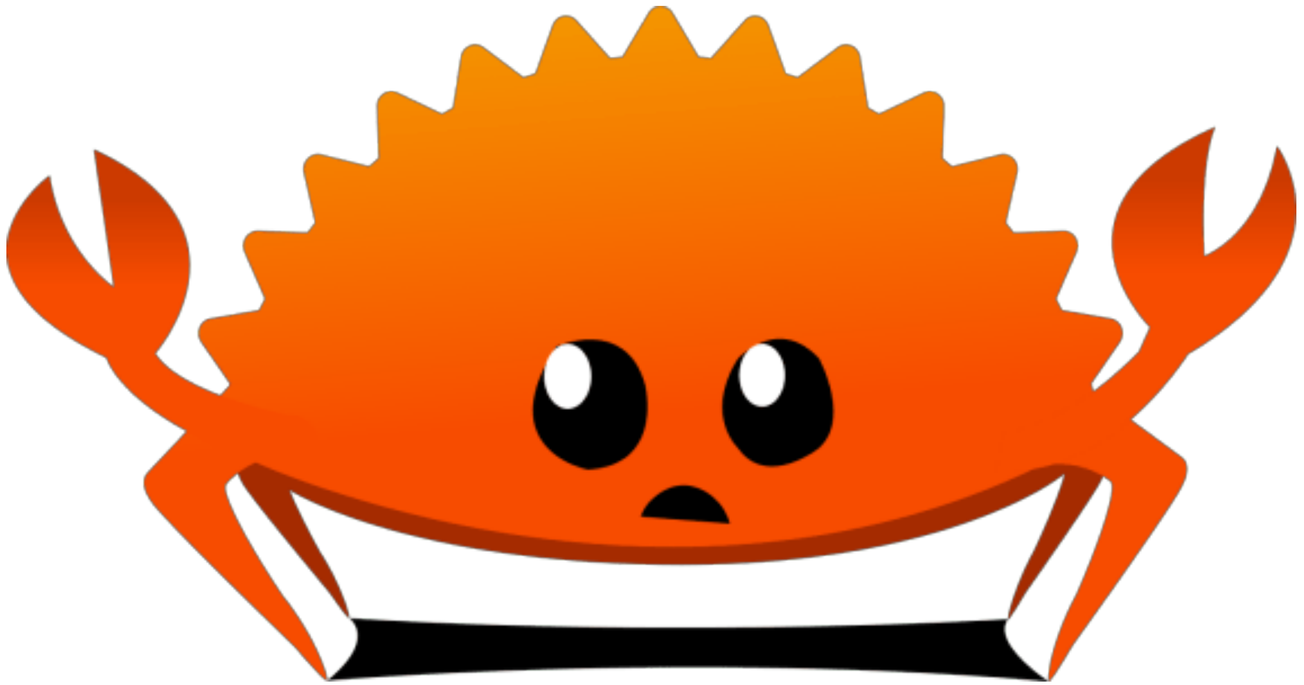
Go

- Straightforward, low complexity
- Easy to optimize code
- Great tooling

Scala

- Learning curve, but write less code
- JVM language
- Makes you feel smart

Why no Rust?



Links

- https://github.com/jestclout
- https://hachyderm.io/@countingtoten