UDACITY

Sair

‹ Return to "Blockchain Developer" in the classroom

DISCUSS ON STUDENT HUB

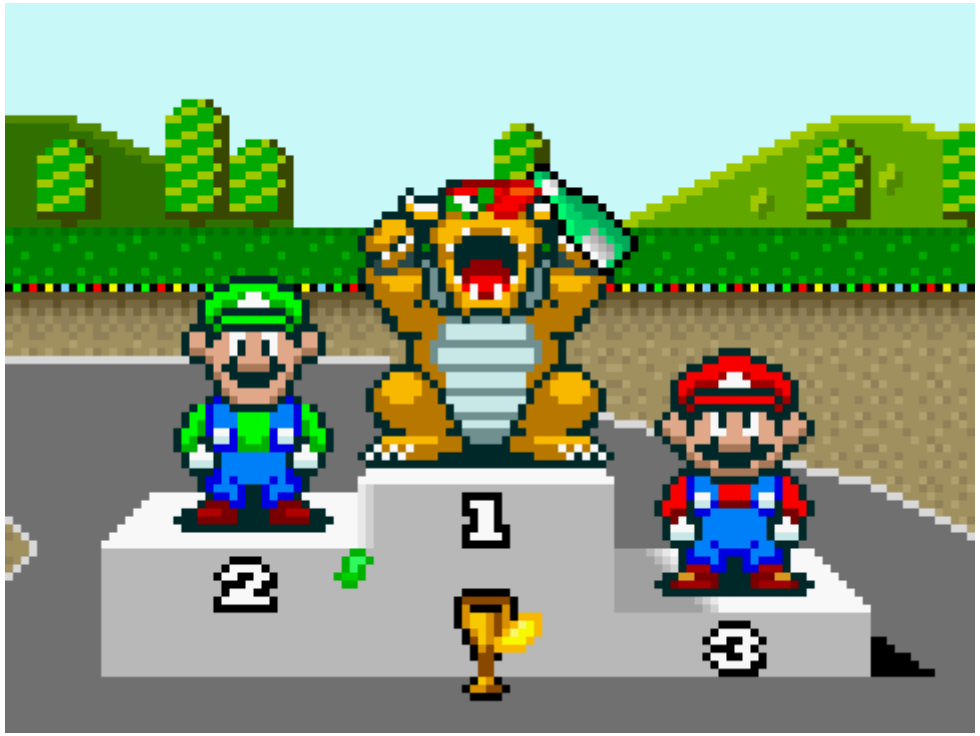
FlightSurety

REVISÃO | REVISÃO DE CÓDIGO 5 | HISTORY

Meets Specifications

You made it!! Congratulations!! I'm very proud of you, and you should be as well!

Avalie esta revisão



I wish you the best in your career and once again congratulations! 🎉

- If you feel like I can be more helpful to you and other students in any manner, please use the feedback session to drop me a hint about it.

Regards,
Daniel

Separation of Concerns, Operational Control and “Fail Fast”

Smart Contract code is separated into multiple contracts:

1) FlightSuretyData.sol for data persistence
2) FlightSuretyApp.sol for app logic and oracles code

Avalie esta revisão

The project is modularized as expected. The `FlightSuretyApp.sol` file controls the logic and oracles and the `FlighrSuretyData.sol` provides data persistence interfaces.

A Dapp client has been created and is used for triggering contract calls. Client can be launched with "npm run dapp" and is available at http://localhost:8000

Specific contract calls:

1) Passenger can purchase insurance for flight
2) Trigger contract to request flight status update

Running the client

- The frontend application starts succesfully and is being served through port 8000 upon a `npm run dapp` command execution

Dapp functionalities

- User can request flight status using the dapp
- User can buy insurance for a given flight

A server app has been created for simulating oracle behavior. Server can be launched with "npm run server"

The backend application starts succesfully upon a `npm run server` command execution

Students has implemented operational status control.

Operational control is implemented through a `modifier` and it is being used correctly on sensitive functions. 👍

Avalie esta revisão

Contract functions “fail fast” by having a majority of “require()” calls at the beginning of function body

A good amount of `require()` 's are being implemented and they do make sense within the application context.

Airlines

First airline is registered when contract is deployed.

The first airline is being deployed upon the contract deploy through a smart contract `2_deploy_contract.js` hook. Great job! 👍

Only existing airline may register a new airline until there are at least four airlines registered

Demonstrated either with Truffle test or by making call from client Dapp

There is a test case available on the `test/flightSurety.js` file which covers the Multiparty consensus logic validation.

Registration of fifth and subsequent airlines requires multi-party consensus of 50% of registered airlines

Demonstrated either with Truffle test or by making call from client Dapp

There is a test case that validates the funding logic.

Airline can be registered, but does not participate in contract until it submits funding of 10 ether

Avalie esta revisão

Demonstrated either with Truffle test or by making call from client Dapp

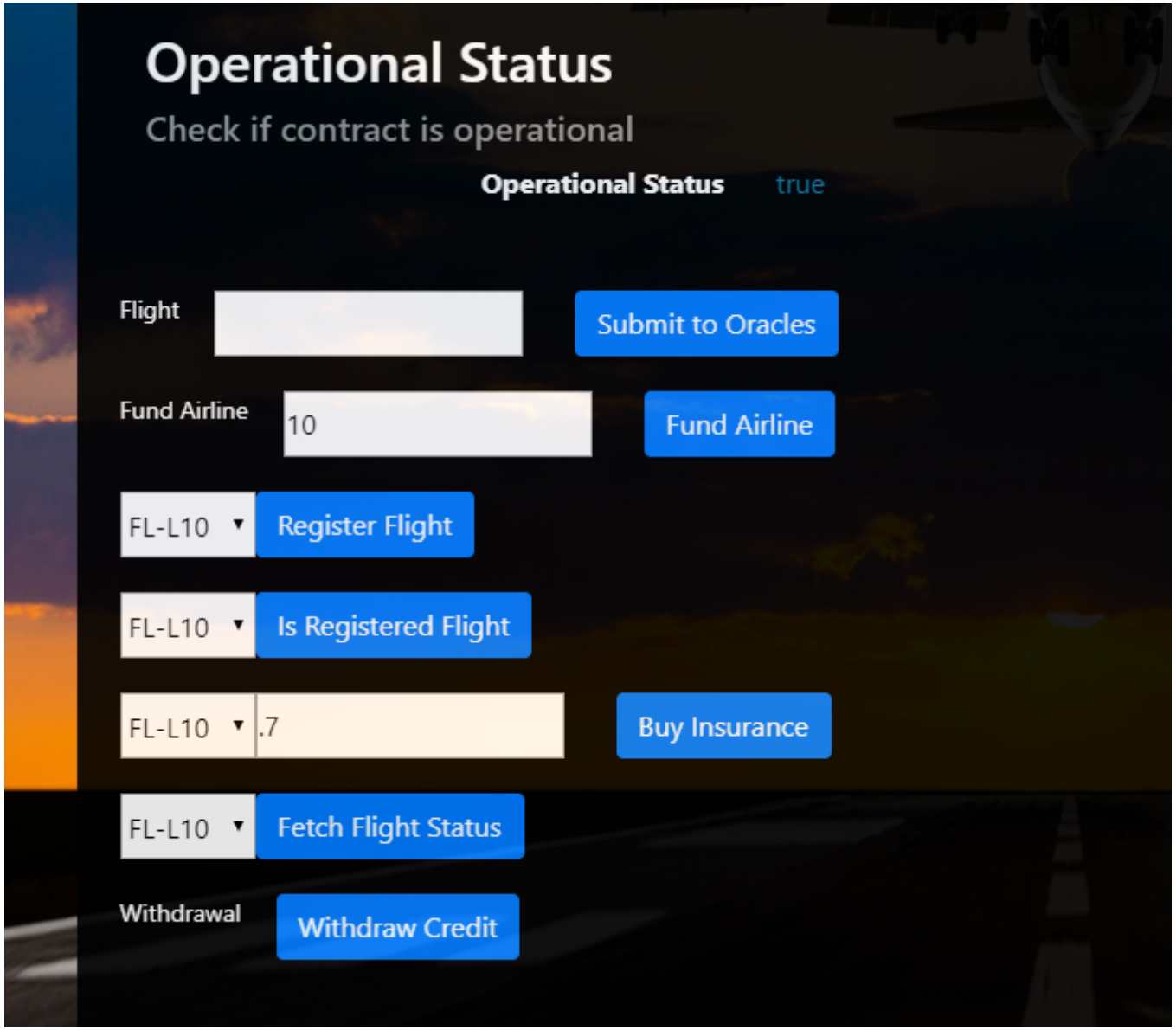
The flight numbers list feature is available on the Dapp.

Passengers

Passengers can choose from a fixed list of flight numbers and departure that are defined in the Dapp client

The flight numbers list feature is available on the Dapp.

Avalie esta revisão



Passengers may pay up to 1 ether for purchasing flight insurance.

This logic is validated through a test case available on `test/flightSurety.js`

Avalie esta revisão

If flight is delayed due to airline fault, passenger receives credit of 1.5X the amount they paid

This logic is validated through a test case available on `test/flightSurety.js`

Passenger can withdraw any funds owed to them as a result of receiving credit for insurance payout

👍

Insurance payouts are not sent directly to passenger's wallet

Oracles (Server App)

Oracle functionality is implemented in the server app.

Oracles are being implemented

Upon startup, 20+ oracles are registered and their assigned indexes are persisted in memory

20 Oracles are being registered upon a server startup

Update flight status requests from client Dapp result in OracleRequest event emitted by Smart Contract that is captured by server (displays on console and handled in code)

Events are being handled, great job

Avalie esta revisão

Server will loop through all registered oracles, identify those oracles for which the OracleRequest event applies, and respond by calling into FlightSuretyApp contract with random status code of Unknown (0), On Time (10) or Late Airline (20), Late Weather (30), Late Technical (40), or Late Other (50)	
	All status case are implemented and a random status is being assigned each time

BAIXAR PROJETO

5 COMENTÁRIOS DA REVISÃO DE CÓDIGO ›

RETORNAR

Avalie esta revisão