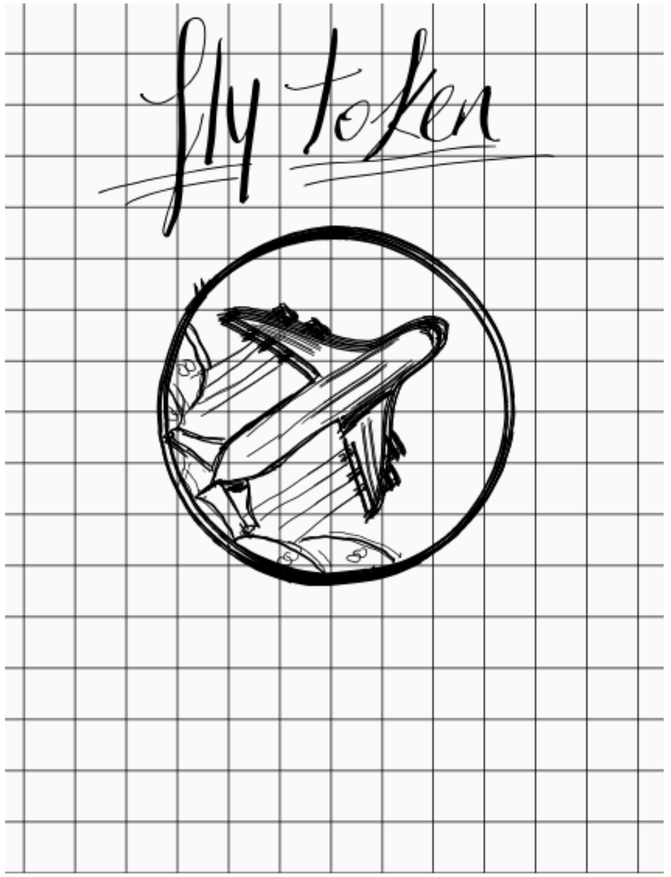
Future of Airline Travel

Combining AI and smart contracts to expedite secured entry



Presented March 3, 2022 by Team 4 (Sumeet Vaidya, Pat Beeson, Bipasha Goswami, Hu TheDesigner, Bill Alford)

Executive Summary

Overview
Bottleneck interference during boarding is the cause of flight delay and increased turnaround times. We propose here a system to expedite the boarding process by enabling contact free entry via facial recognition and a smart contract.

ML
We developed a system using Biometric authentication as a form of identification and access control. The passenger checks into a flight via a distributed application by taking either a live photograph or video of themselves and comparing to officially verifiable identification, either a drivers license or for international travel a passport.

IPFS
During the checkin flow the passenger uploads their passport-scan to IPFS and submits the IPFS hash to the function to construct their ERC721 Boarding Pass NFT which includes this IPFS hash.

ERC721 Contract
The main design decision for the contract is to use the ERC721 token standard for Non-Fungible to model boarding passes which are distributed to passengers upon successful completion of checkin web-flows. ERC721 NFTs enabled us to token assets with distinctive characteristics. This token standard is perfectly suited to value-assets like flight seats and boarding passes where each boarding pass should be associated with a unique passenger. The benefit of allowing the passenger to hold an ERC721 seat after booking is that it allows the passenger to sell, trade, swap, or give away their seat before checkin, if they wish to.

OpenZeppelin
To implement ERC721 NFTs, the Booker program inherits from the OpenZeppelin implementation of the the ERC721 standard.

Hope you enjoy our presentation!

Table of Contents

Approach taken to achieve project goals

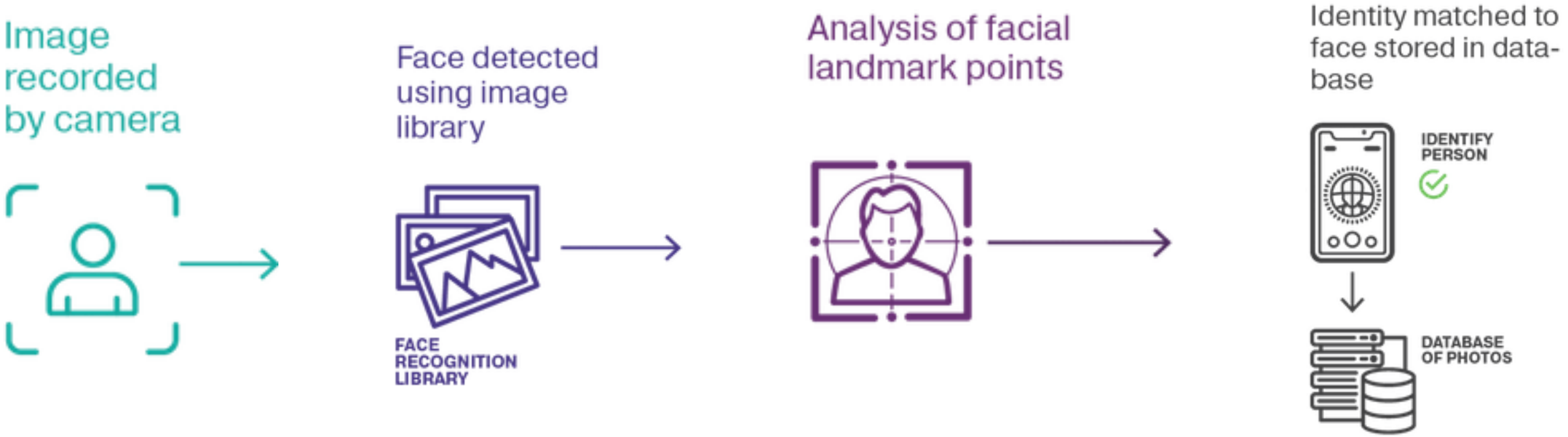
- Description of Facial Recognition process
- Steps taken to build a Distributed Application
- What our smart contract contains
- Project Challenges
- Next Steps given more time
- Questions

Approach to achieving project goals

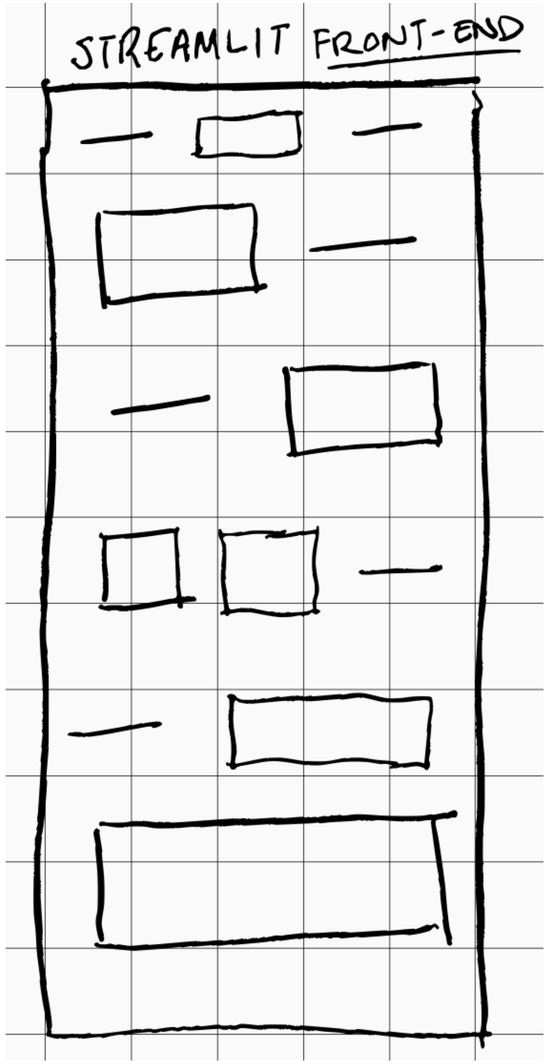
Overview of completed steps

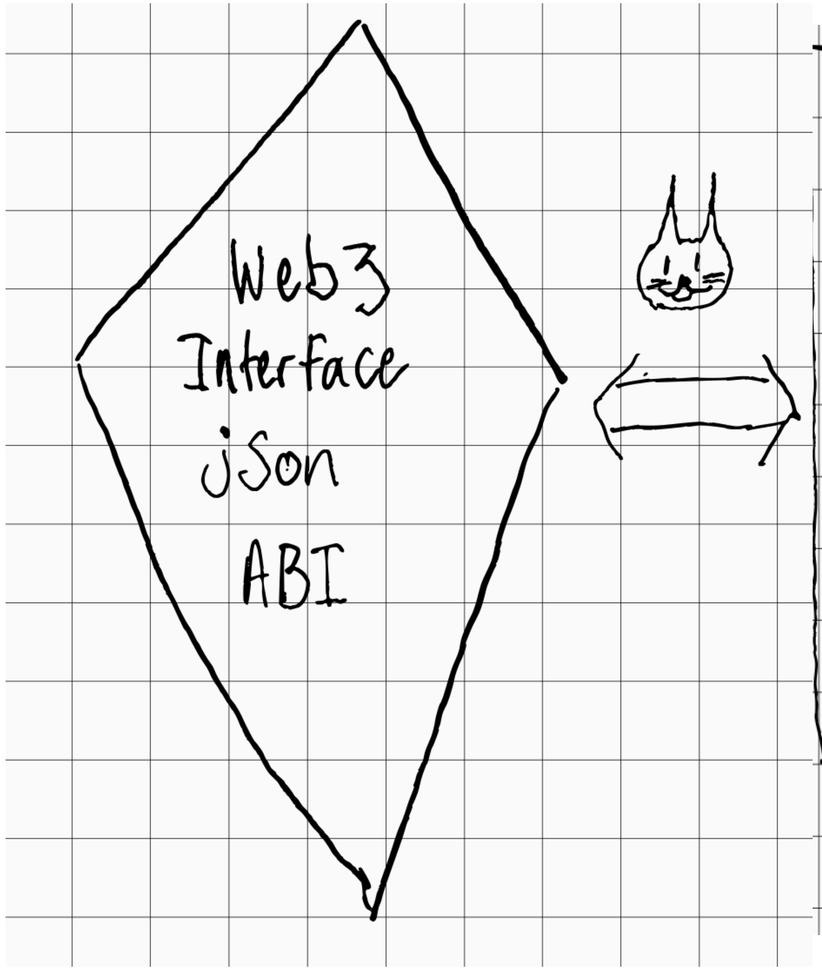
- Build Multi App application in Steamlit and Remix
- Imported the scikit-learn-intelex, opencv, dlib, face_recognition machine learning libraries
- Developed and integrated two integrated web pages, biometric login and boarding pass
- Biometric login allows users to confirm identity by uploading a saved image or using live image detect or webcam video
- Boarding pass app allows passenger to use their wallet to interact with a deployed smart contract to bypass traditional checkin and retrieve boarding pass via the contract
- Images are then compared in the app and recognized images then create smart contract resulting in delivery of boarding pass
- Used IPFS to store official images and deliver ERC721 tokenized image
- ERC721 token can then be traded on NFT marketplace such as OpenSea as a collectable
- Use application to sign into blockchain, register, collect token, display token and receive boarding pass

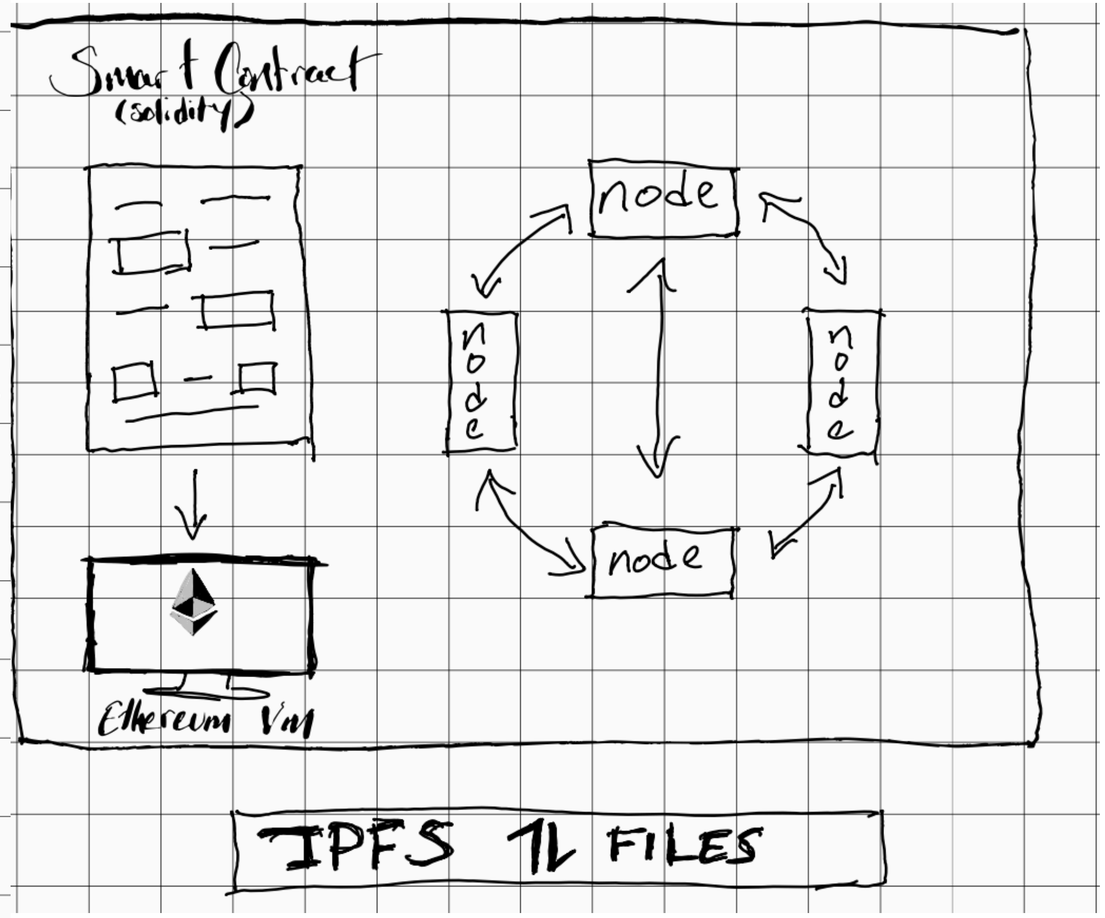
Facial Recognition process



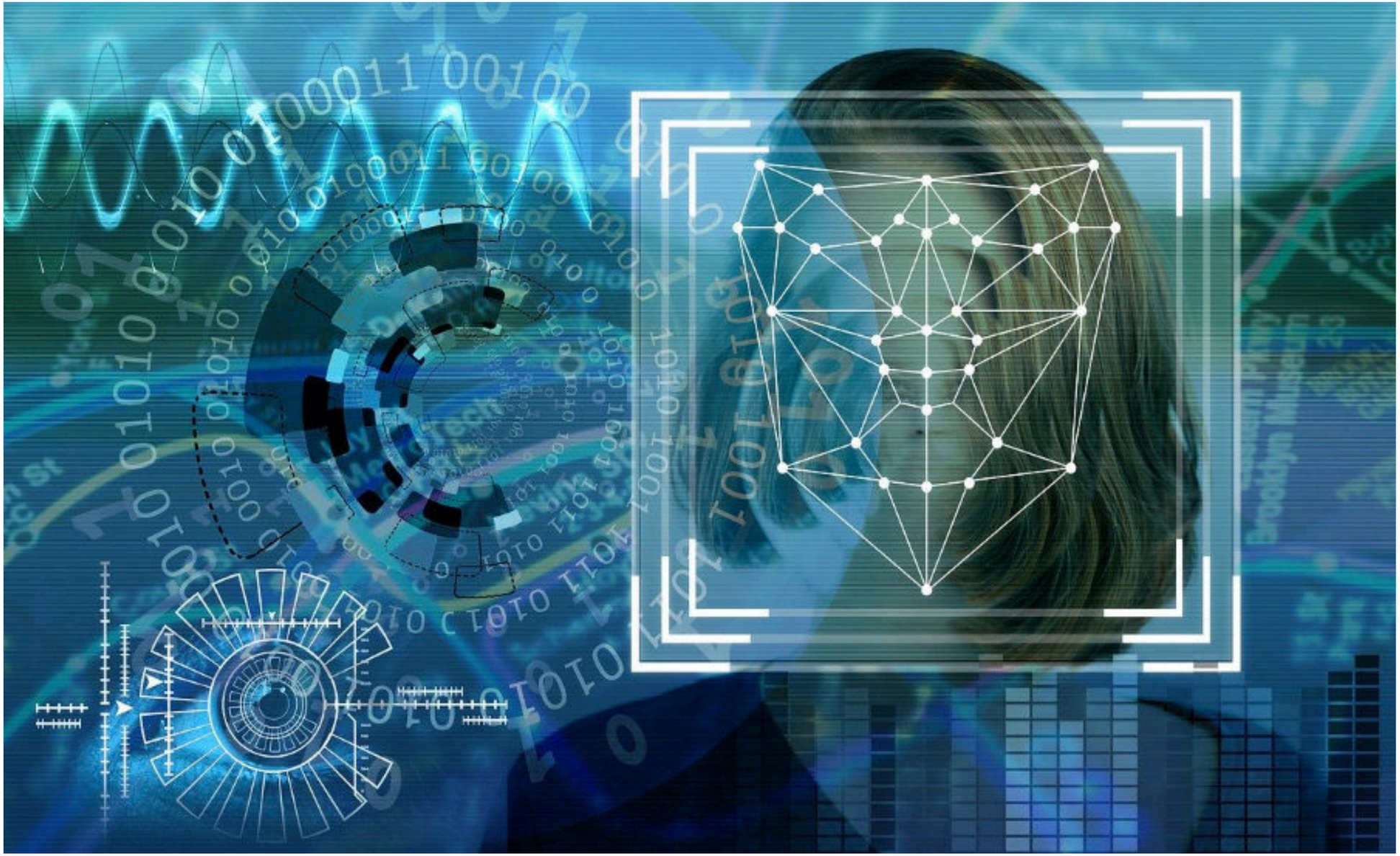
dApp diagram







Demo



Next steps if more time

- Had we went on further, we would have a unique one of a kind NFT collection with different values and rarities that form collectible rare items which hold value with purchase of each ticket.
- The Utility of the ticket would be depleted, but the NFT value would be inherently valued by the community and would have trade-able value.
- Different NFTs would be used

The End

This concludes our presentation.

dAPP configuration

What is a dApp?

A decentralized application (dApp) is an application built on a decentralized network that combines a smart contract and a frontend user interface. On Ethereum, smart contracts are accessible and transparent - like open APIs -so your dapp can even include a smart contract someone else has written

How in the heck do I build one?

First build a dApp back-end contract

- Import code from OpenZeppelin ERC721Full contract
- Build your contract
- Create public functions to be called in your app

Launch a local ganache instance

Import accounts from ganache to MetaMask

Compile and deploy contract in the Remix IDE

- Copy the application binary interface (ABI) to a local JSON file

Build a dApp Front End

- Use Streamlit components and Web3.py library
- Set up the Application
- Setup an environment file named .env
- Create a .json file to contain the content of the compiled contract's ABI file
- Create a app.py file to serve as the main application file

Deploy your application to end users