CheckersVision

Charles Dudley
Ranai Srivastav
Sibhat Yosef
Siddhant Jaju
Yaaseen Basha

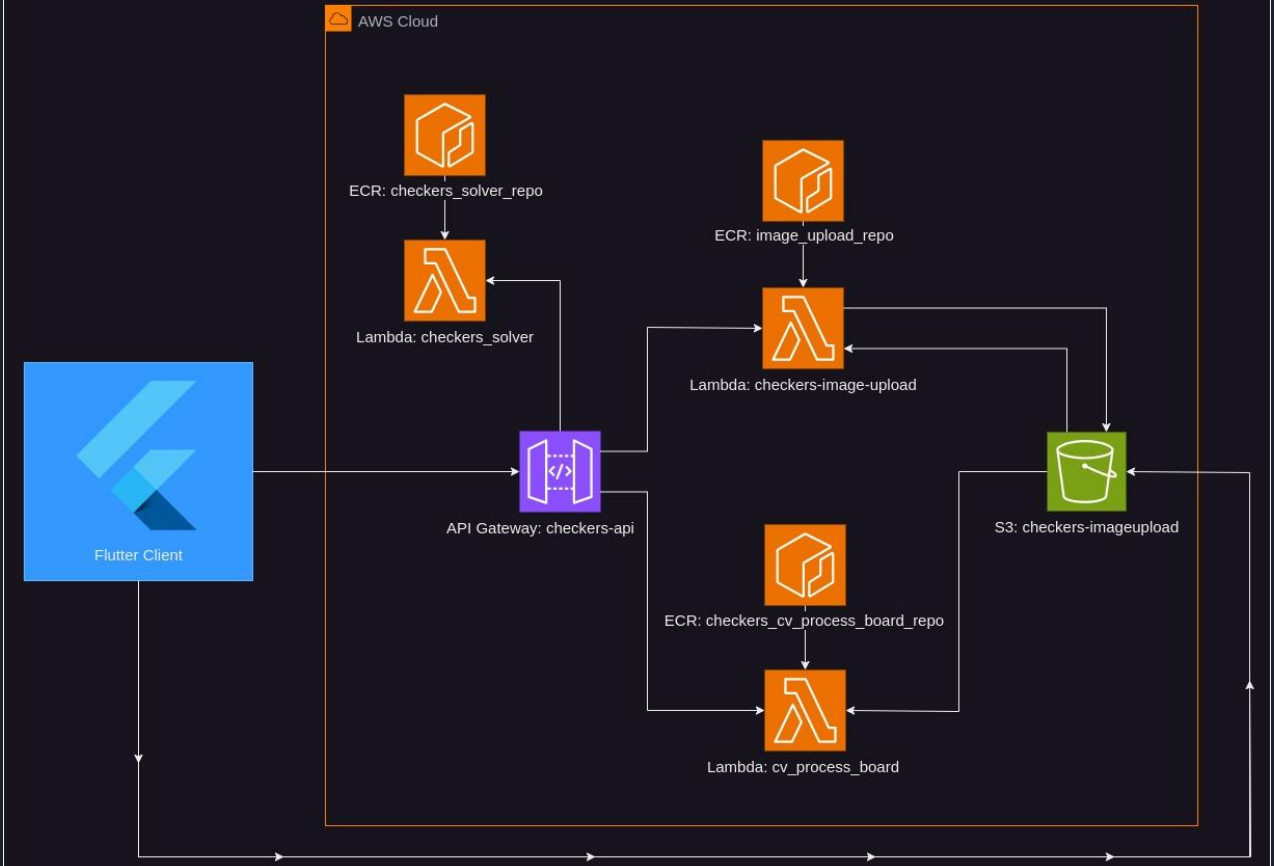
What is CheckersVision?

Recap

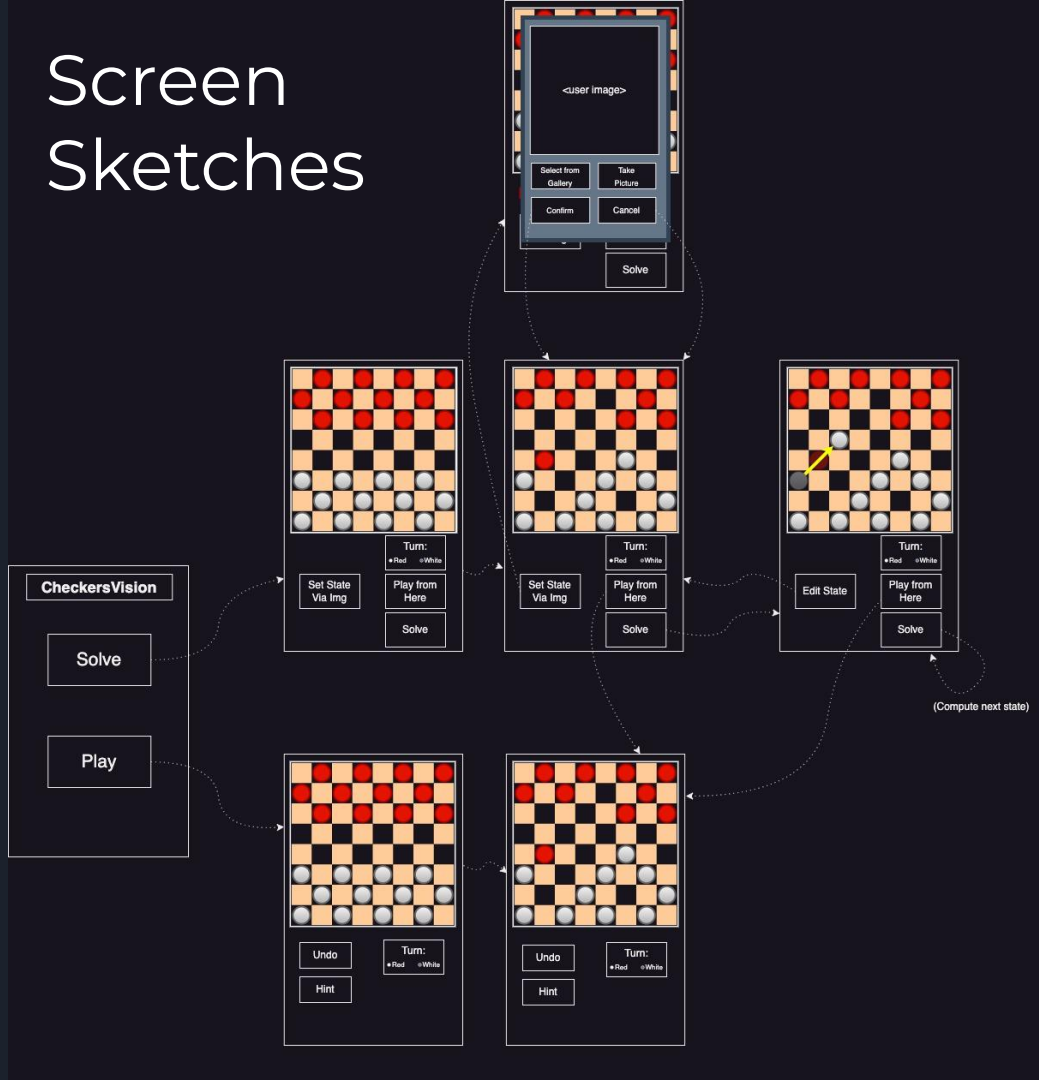
A mobile application where users can:

- Play a complete Checkers game against an AI
- Learn what the next best move is based on:
  - A photo of a Checkers board (Computer Vision)
  - Manual input

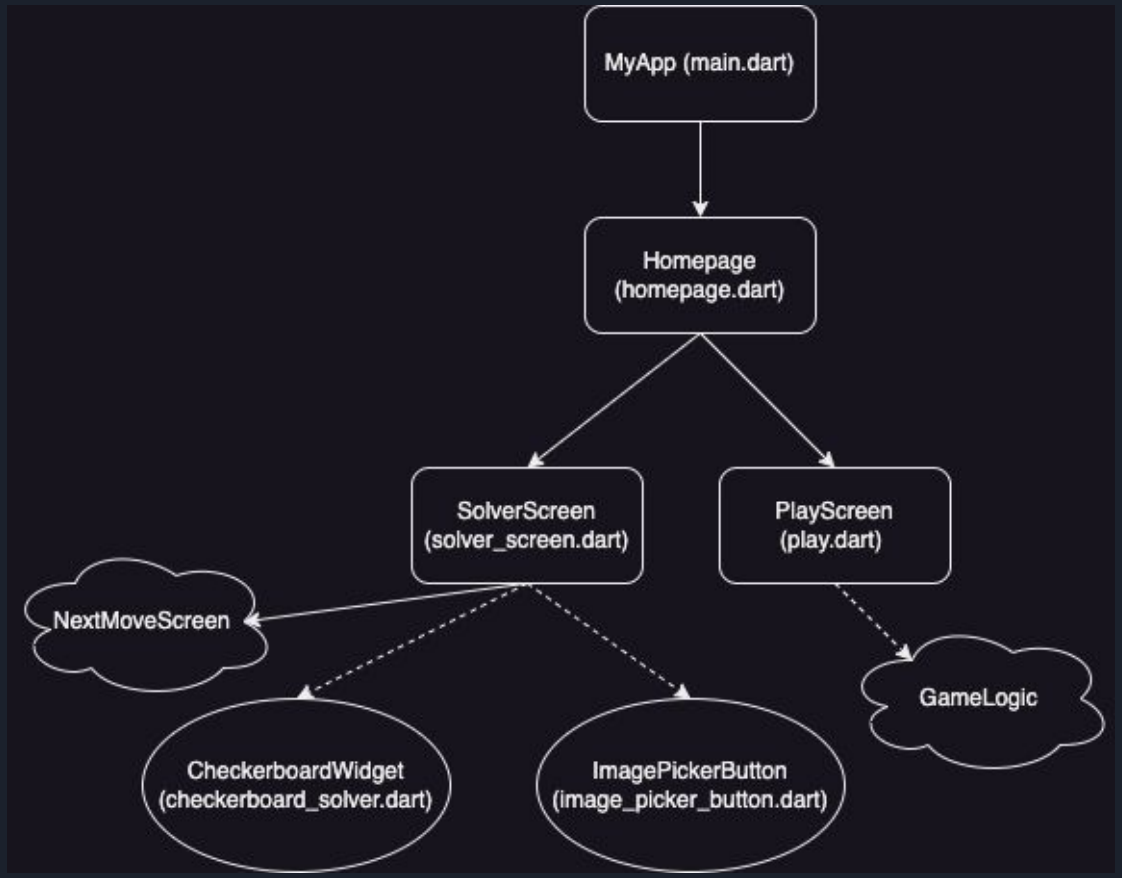
Infrastructure



Screen Sketches



Frontend Format



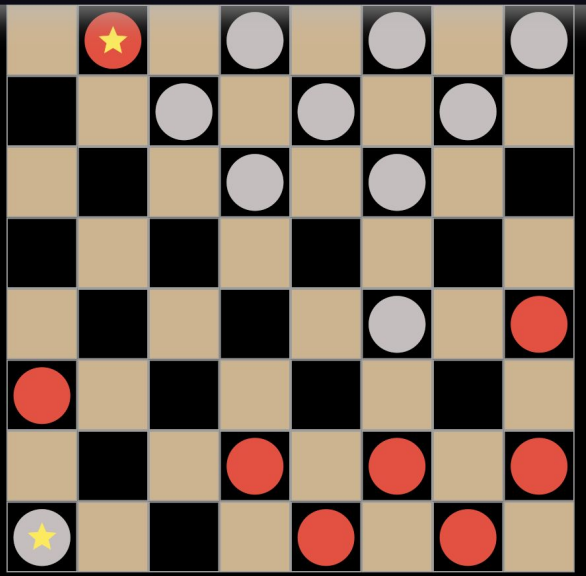
Zooming in...

- Board building logic
- Saved board state representation

```
Saved Board State: [[0, -2, 0, 1, 0, 1, 0, 1], [0,
0, 1, 0, 1, 0, 1, 0], [0, 0, 0, 1, 0, 1, 0, 0], [0,
0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 1, 0, -1], [-
1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, -1, 0, -1, 0, -
1], [2, 0, 0, 0, -1, 0, -1, 0]]
```

```
case CellState.grey:
  //If grey and under limit, upgrade to grey king, else go to empty
  if (currentGreyPieces <= maxPiecesPerColor) {
    board[row][col] = CellState.greyKing;
  } else {
    board[row][col] = CellState.empty;
    currentGreyPieces--;
  }
  break;
```

Checkers Vision

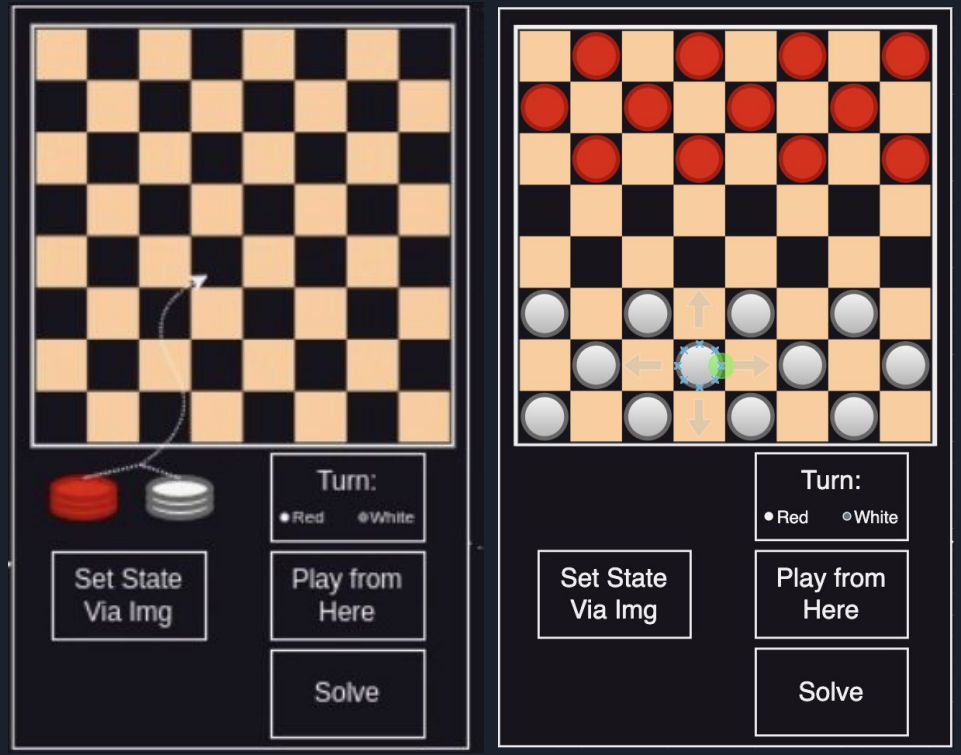


Save Game State

Set State Via Img

Frontend - Challenges

- Designing from scratch and learning the Flutter Framework
- Rework from “Drag and Drop” implementation to “On-Click” changes
- Pausing on certain features until back-end progress



Frontend - Remaining Work

- State management to traverse between the “Solver” and “Play Checkers” screen.
- A screen to display a Checkers board with the next best state when given a jSON object
- Full implementation of “Play Checkers”

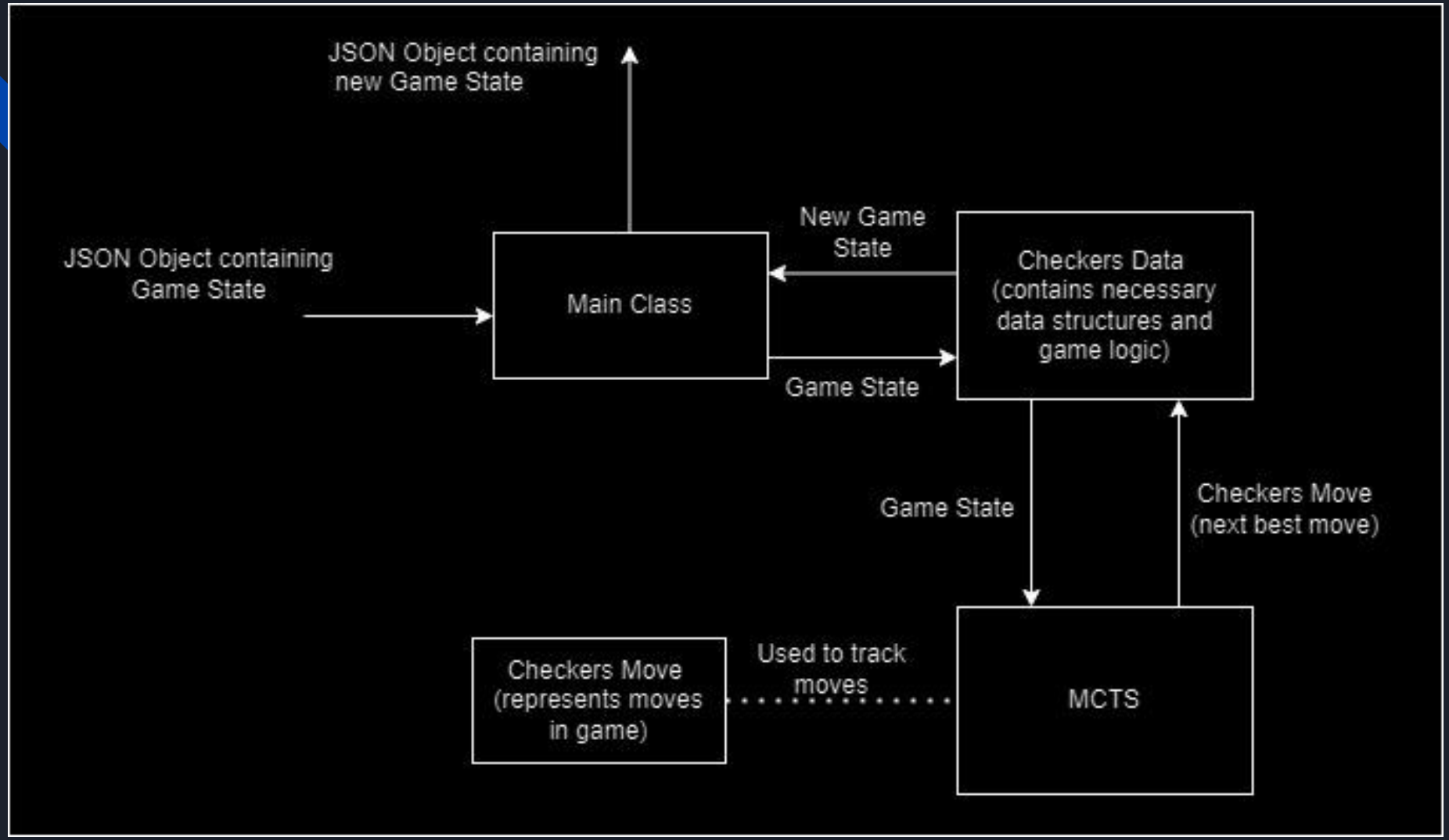
Checkers Engine

What is our Checkers Engine?

- Backend implementation - has the design and logic for the Checkers Game.

What are the components?

- **Checkers Data** : Contains all the necessary data structures needed for the checkers engine.
- **Checkers Move**: Contains the logic of how the pieces will move.
- **MCTS:** Main search algorithm used to determine the next best/most optimal move for the user.

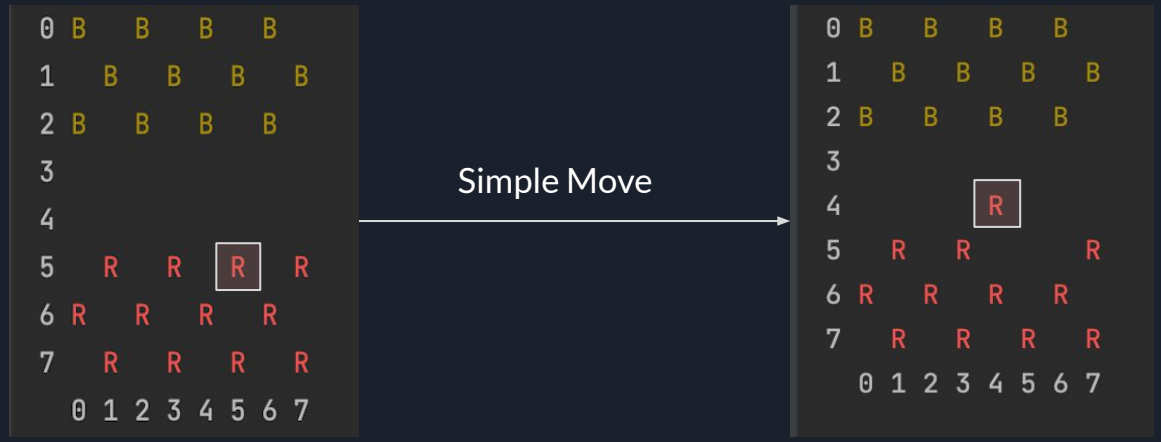

JSON Object containing new Game State
JSON Object containing Game State
Main Class
New Game State
Game State
Checkers Data (contains necessary data structures and game logic)
Game State
Checkers Move (next best move)
Checkers Move (represents moves in game)
Used to track moves
MCTS

Testing Move Application

```java
@Test
public void testApplySimpleMove() {
    System.out.println(board);
    CheckersMove simpleMove = new CheckersMove( r1: 5,  c1: 5,  r2: 4,  c2: 4);
    board.makeMove(simpleMove);
    System.out.println(board);

    assertEquals( message: "Destination should be occupied by RED", CheckersData.RED, board.pieceAt( row: 4,  col: 4));
    assertEquals( message: "Original position should be EMPTY", CheckersData.EMPTY, board.pieceAt( row: 5,  col: 5));
}
```

Board Representation



Initial State
(5,5)

State after move
(4,4)

Testing the Checkers Engine

- Lack of visual feedback on the backend
- Testing ensures implementation correctness independent of frontend
- Testing has already been successful in detecting implementation bugs
- GOALS:
  - 60% average code coverage over Checkers Engine
  - 80% code coverage for crucial methods

Checkers Engine- Implementation Challenges

- Designing the data structures for the Checkers Engine
- Implementing the ‘makeMove()’ for MCTS
- Testing MCTS

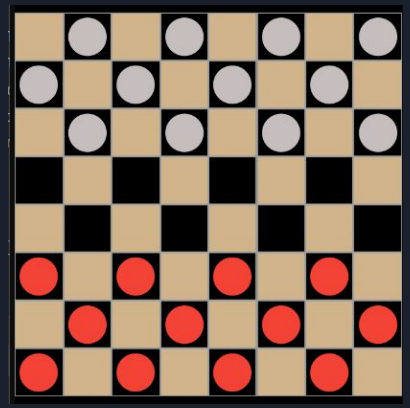
Computer Vision Design Overview

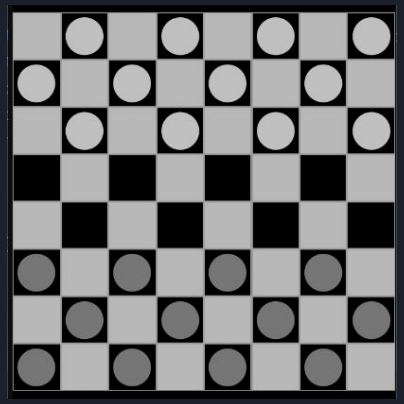
Detecting pieces by what the color is at the midpoint of each square instead of detecting circles

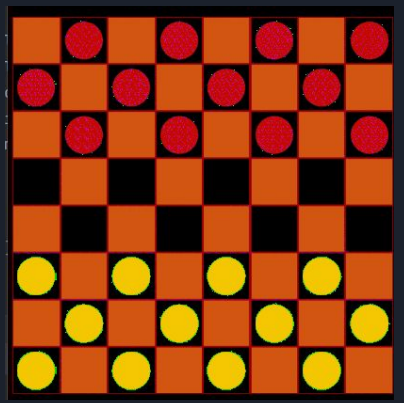
Load Image

```
# Loads an image
src = cv.imread('img/frontend.png')
srcGray = cv.cvtColor(src, cv.COLOR_BGR2GRAY)
src_hsv = cv.cvtColor(src, cv.COLOR_BGR2HSV)
# Check if image is loaded fine
```

Need to load image and convert to Grayscale and HSV





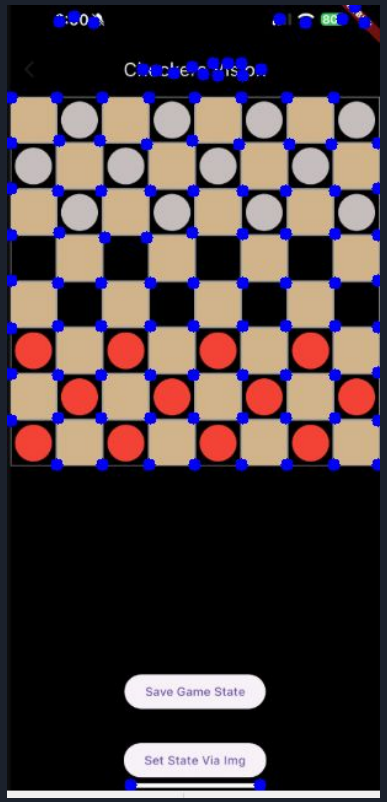


Board Detection

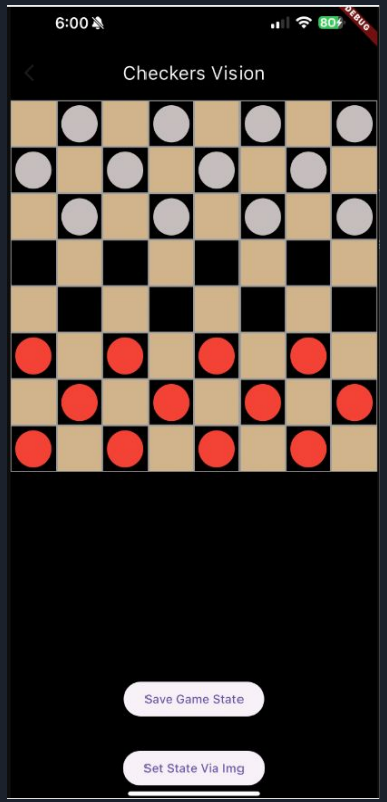
Many ways to go about this:

- goodFeaturesToTrack
- findChessboardCorners
- Harris Corner Detection
- Hough Line Detection

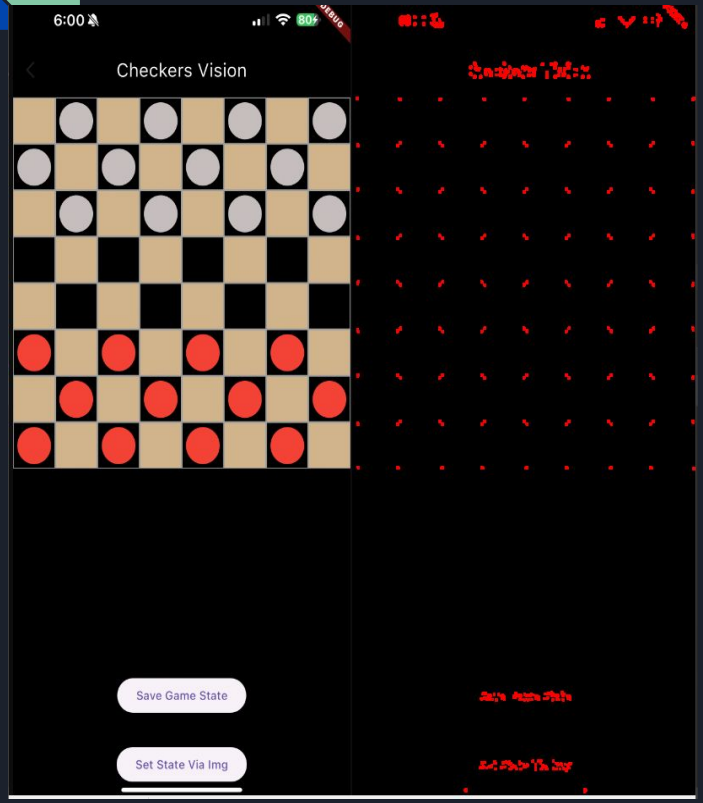
goodFeaturesToTrack



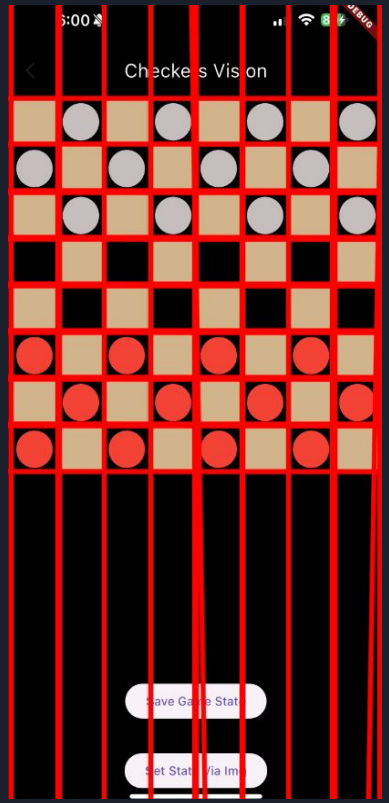
findChessboardCorners



Harris Corner Detection

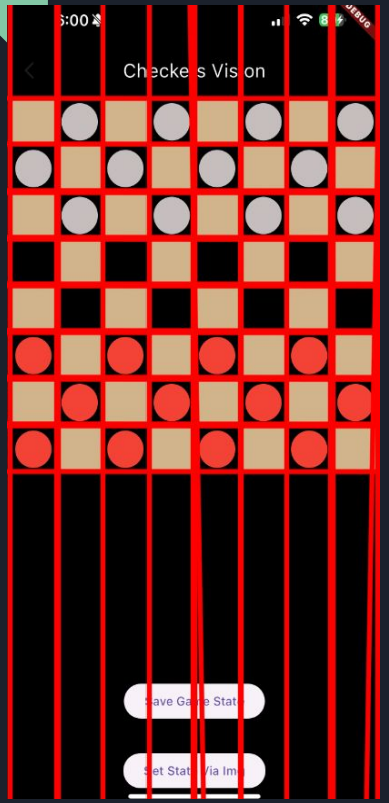


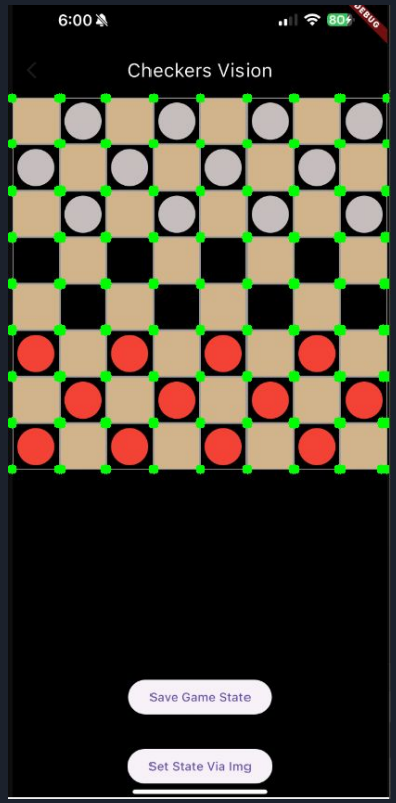
Hough Line Detection



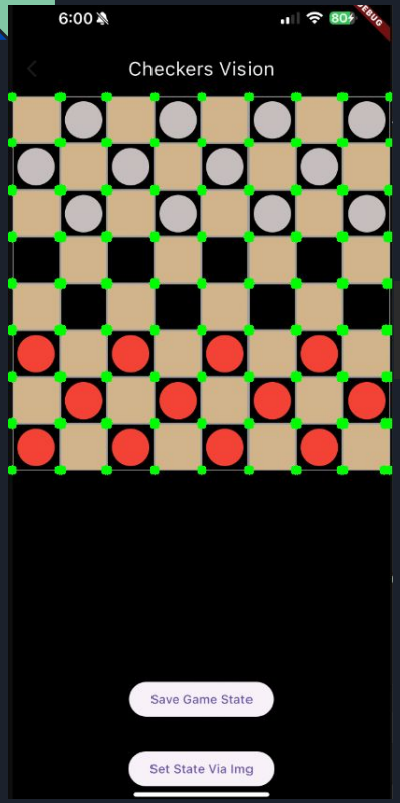
Corner Detection

Find corners using intersection of lines



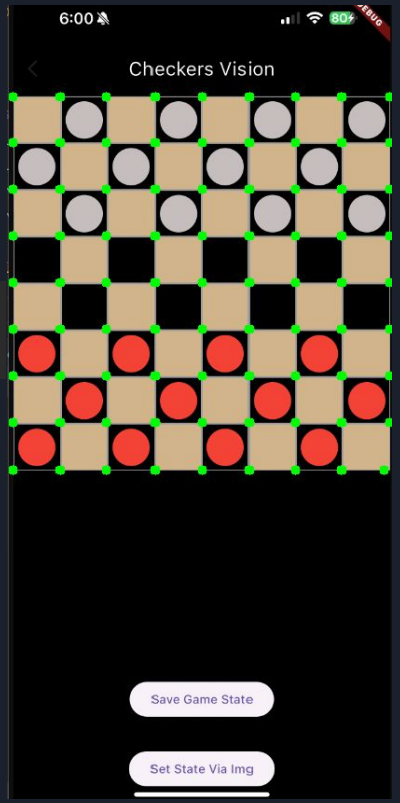


Anomaly Processing



Remove duplicate corners by removing points that seem too close to one another

```
if (col - prev_col) < threshold and (col - prev_col) != 0:
    sorted_corners.remove(corner)
else:
```



Piece Detection

With corners, create squares

```
class Square:
    def __init__(self, top_left, bottom_left, top_right, bottom_right) -> None:
        self.top_lft = top_left
        self.bot_lft = bottom_left
        self.top_rt = top_right
        self.bot_rt = bottom_right

    def get_midpoint(self):
        x = (self.top_rt[0] + self.top_lft[0]) // 2
        y = (self.bot_lft[1] + self.top_lft[1]) // 2
        return x, y
```

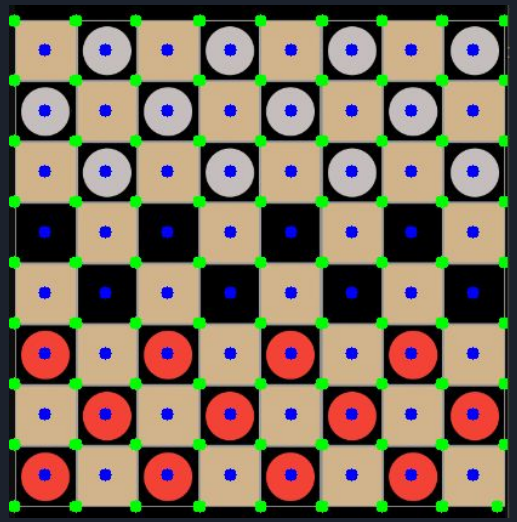
Piece Detection

Sort corners by y-coordinate to group rows

Then sort by x-coordinate to get board

```
Row 1 -: [(4, 101), (54, 101), (103, 101), (155, 101), (205, 101), (255, 101), (304, 101), (353, 101), (404, 101)]
Row 2 -: [(4, 150), (54, 150), (103, 150), (155, 150), (205, 150), (255, 150), (304, 150), (353, 150), (404, 150)]
Row 3 -: [(4, 200), (54, 200), (103, 200), (155, 200), (205, 200), (255, 200), (304, 200), (353, 200), (404, 200)]
Row 4 -: [(4, 250), (54, 250), (103, 250), (155, 250), (205, 250), (255, 250), (304, 250), (353, 250), (404, 250)]
Row 5 -: [(4, 300), (54, 300), (103, 300), (155, 300), (205, 300), (255, 300), (304, 300), (353, 300), (404, 300)]
Row 6 -: [(4, 350), (54, 350), (103, 350), (155, 350), (205, 350), (255, 350), (304, 350), (353, 350), (404, 350)]
Row 7 -: [(4, 400), (54, 400), (103, 400), (155, 400), (205, 400), (255, 400), (304, 400), (353, 400), (404, 400)]
Row 8 -: [(4, 450), (54, 450), (103, 450), (155, 450), (205, 450), (255, 450), (304, 450), (353, 450), (404, 450)]
Row 9 -: [(4, 501), (54, 501), (103, 501), (155, 501), (205, 501), (255, 501), (304, 501), (353, 501), (398, 501)]
```

Kinged Piece



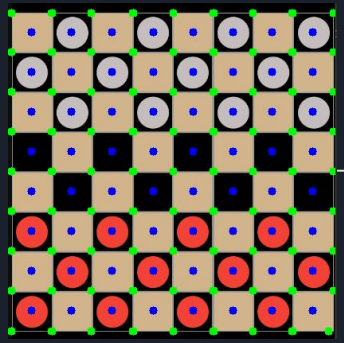
Piece Detection

- **HSV** vs RGB
- Hough circles vs **Color at midpoint**
- Kinged pieces

```
Square 1 - HSV Values: [ 18  87 209], Color: Empty Peach
Square 2 - HSV Values: [  0   9 197], Color: White
Square 3 - HSV Values: [ 17  85 209], Color: Empty Peach
Square 4 - HSV Values: [  0  12 198], Color: White
Square 5 - HSV Values: [ 17  85 209], Color: Empty Peach
Square 6 - HSV Values: [  0  12 198], Color: White
Square 7 - HSV Values: [ 17  86 210], Color: Empty Peach
Square 8 - HSV Values: [  0  10 197], Color: White
Square 9 - HSV Values: [  0   9 197], Color: White
```

Setting Up Return Data Structure

With color of pieces identified, create board representation to send it back to the frontend



-2 is RED King
-1 is RED Normal
0 is EMPTY Square
+1 is BLACK Normal
+2 is BLACK King

```
0   1   0   1   0   1   0   1
1   0   1   0   1   0   1   0
0   1   0   1   0   1   0   1
0   0   0   0   0   0   0   0
0   0   0   0   0   0   0   0
-1  0   -1  0   -1  0   -1  0
0   -1  0   -1  0   -1  0   -1
-1  0   -1  0   -1  0   -1  0
```

CV - Challenges and Remaining Work

Challenges:

- Handling duplicate corner detections
- Ignoring non-board items
- Piece color detection
- Implementation of identifying kinged pieces

```
for index in yellow_indices:
    square = squares[index]
    x, y = square.get_midpoint()
    y = y - 15
    color_hsv = src_hsv[y, x]
    h, s, v = color_hsv
    if h < 5 and s < 15:
        id_colors[index] = "White King"
    elif 190 < s < 210 and 240 < v < 260:
        id_colors[index] = "Red King"
    else:
        id_colors[index] = "Unknown"
```

Remaining work:

- Robust detection
  - Detecting a bad screenshot
  - Different boards with different colors
  - Anomaly processing
  - Kinged pieces

AWS Architecture

- **Serverless** architecture hosted on AWS
- **AWS Services used**
  - S3 (Simple Storage Service)
  - Lambda
  - API Gateway
  - ECR (Elastic Container Registry)
  - CloudWatch

API Gateway

Lambda

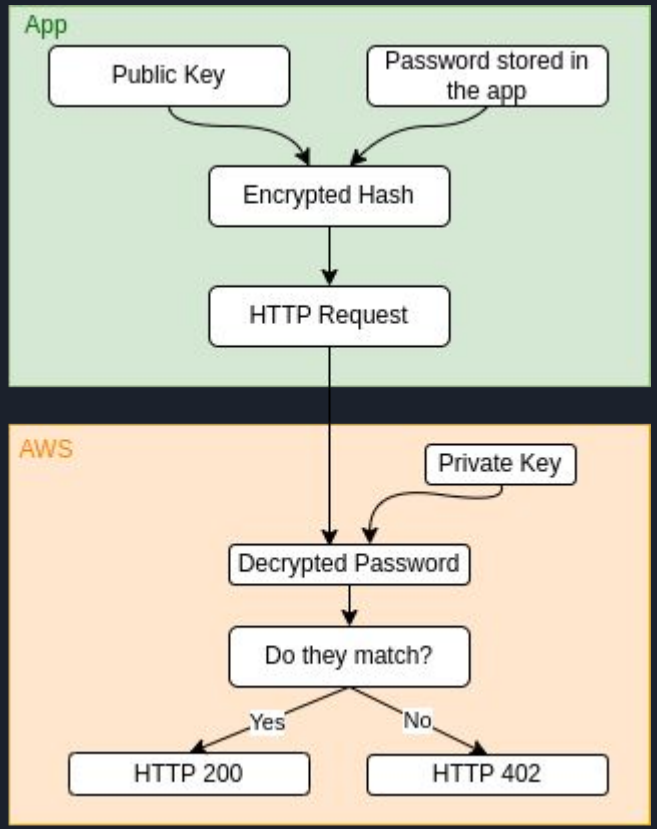
S3

ECR

CloudWatch

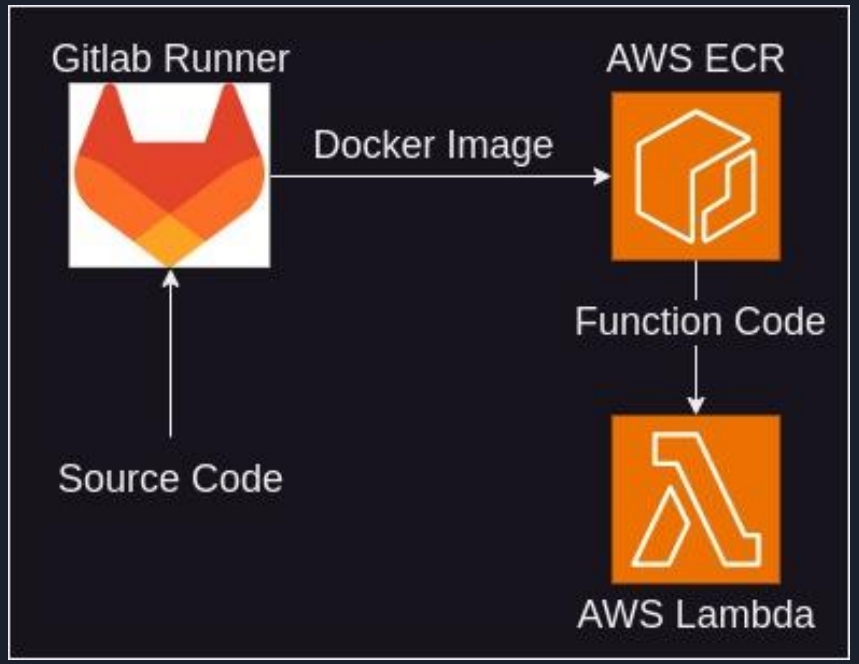
Authentication

- Prevents unauthorized usage of
  - Our Lambda functions
    - CV
    - Pre-signed URL generation
    - Checkers Engine
  - S3 Bucket
- Public and private certificates to ensure only authorized resource access



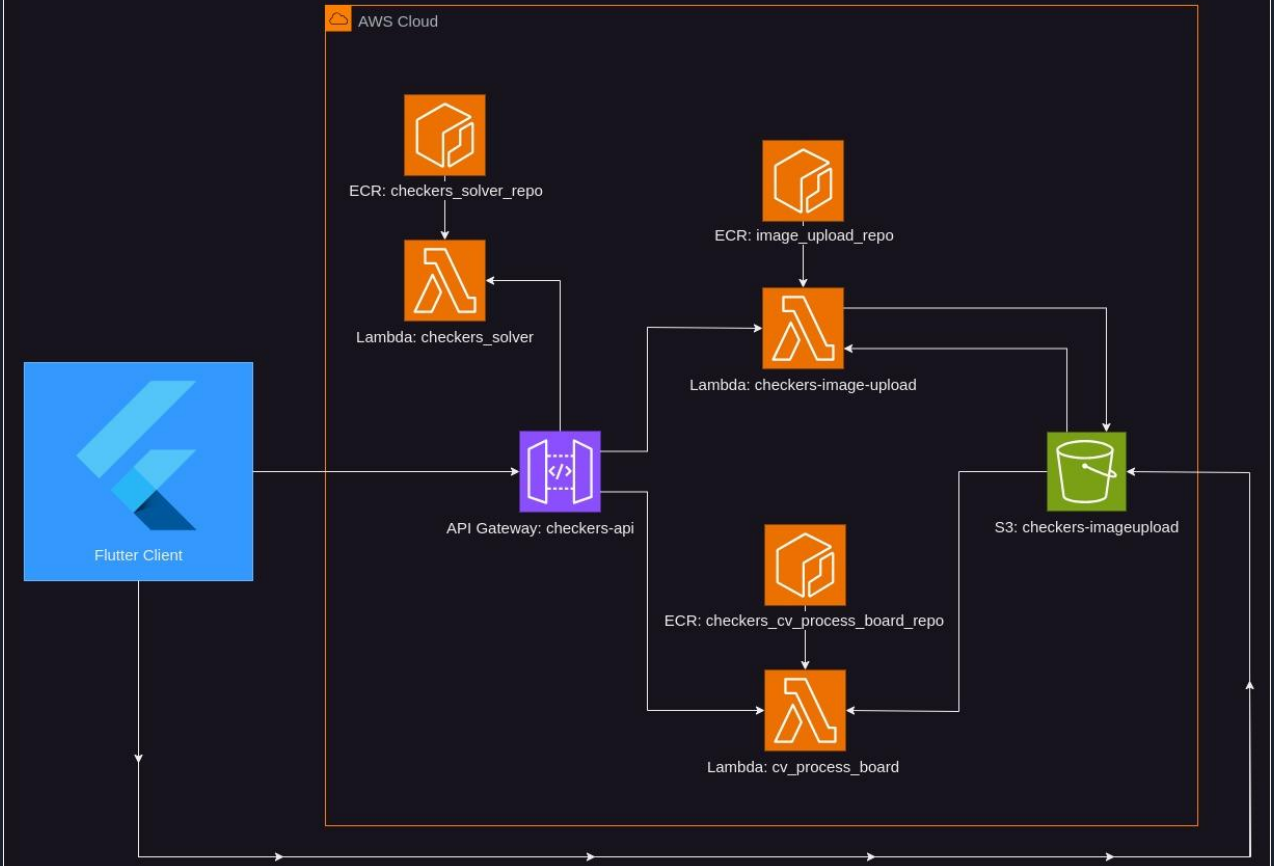
CI/CD with Gitlab Runners

- **Lambdas pull function code from ECR**
  - Package dependencies
  - Facilitate more efficient source control & development
- **What our Runners do:**
  - Clone our project (at most recent commit)
  - Build docker image
  - Tag image as “latest”
  - Push image to AWS ECR
  - Update Lambdas to pull from new “latest” image

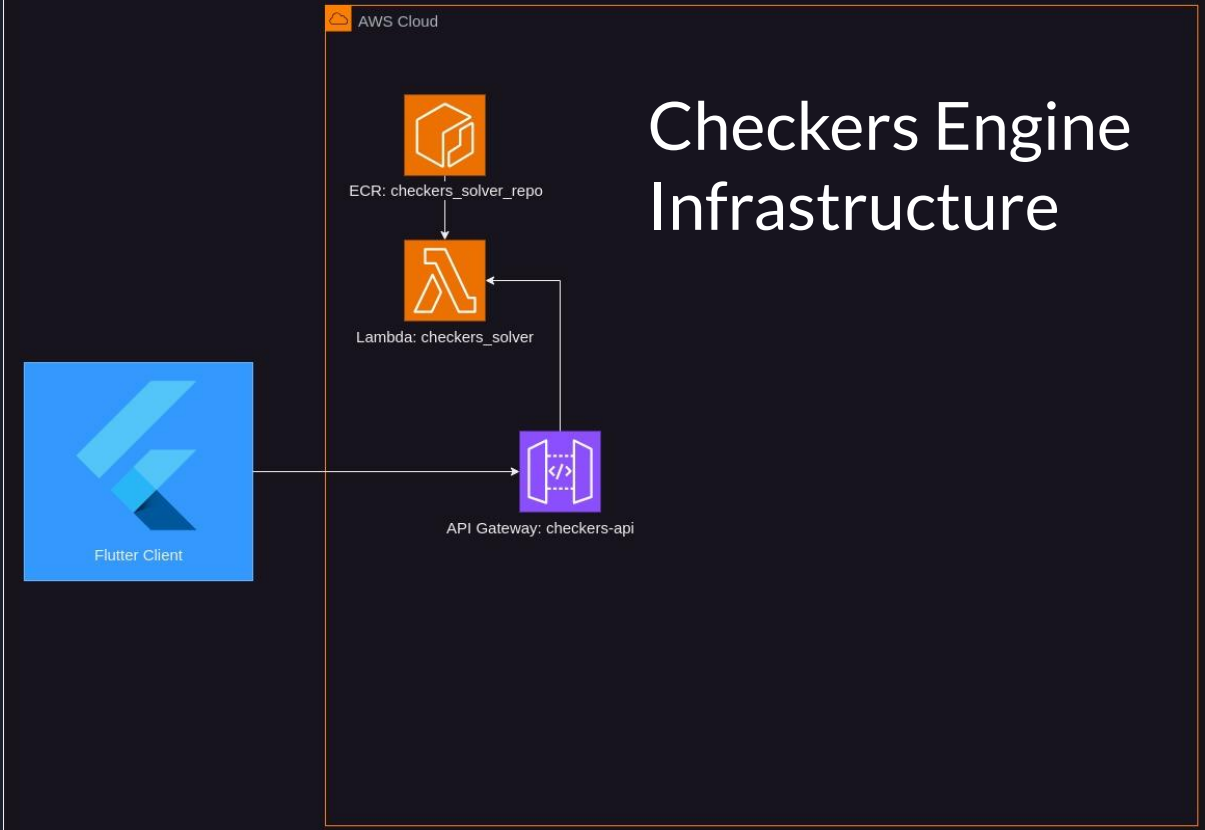


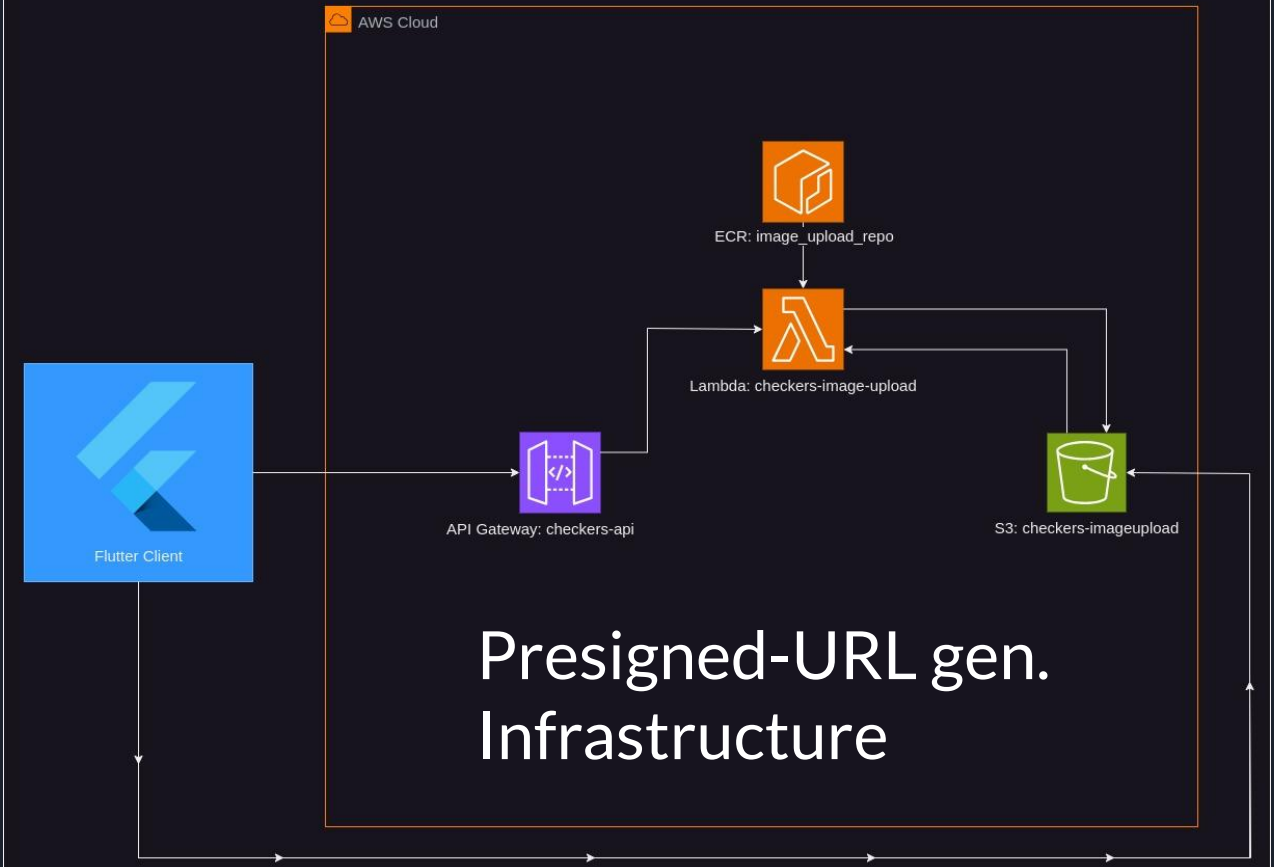
CI/CD Pipeline

Infrastructure



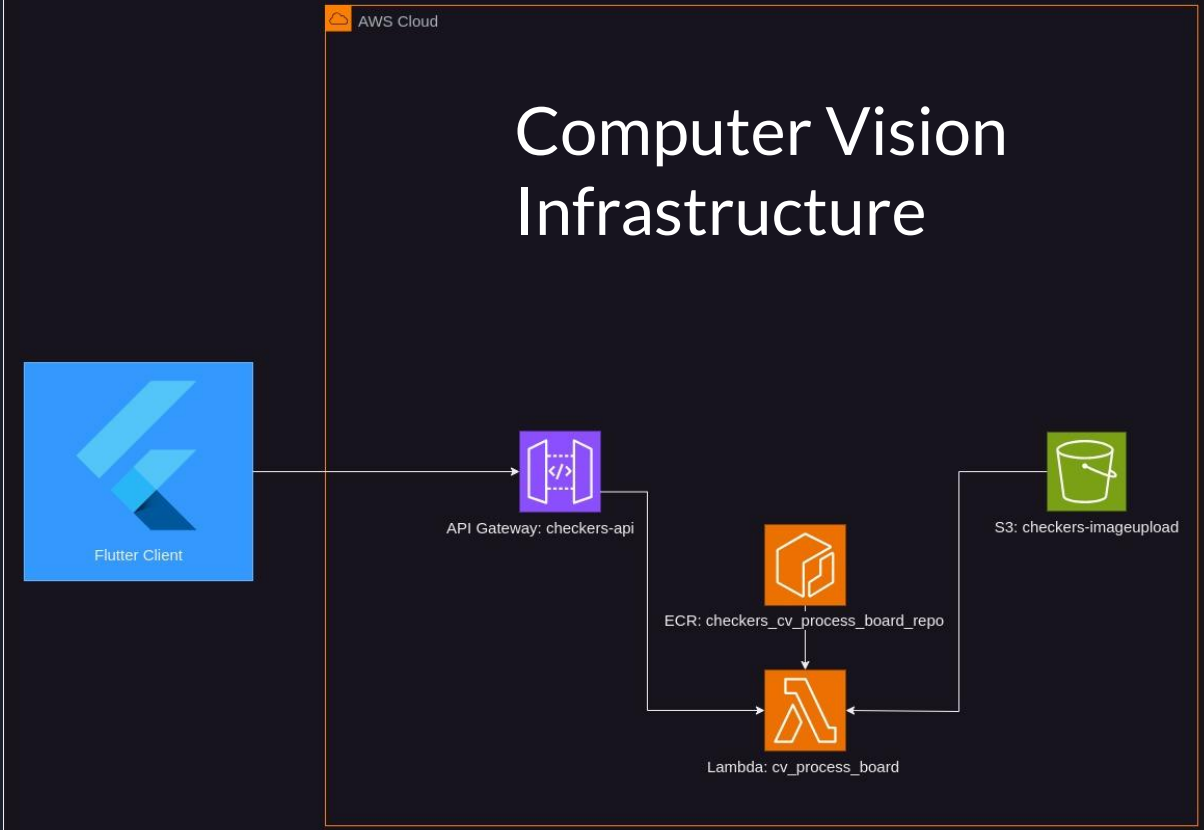
Checkers Engine Infrastructure





Presigned-URL gen. Infrastructure

Computer Vision Infrastructure



Infrastructure - Challenges

- Complex configuration for each AWS Service
- Authentication implementation challenges
- Lambda code editor lacks features
  - No version control
  - No capability for environment setup
- Presigned URL generation:

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Error>
   <Code>InvalidRequest</Code>
   <Message> The authorization mechanism you have provided is not supported. Please use AWS4-HMAC-SHA256. </Message>
   <RequestId>9Y1MEHD65VRPZK8M</RequestId>
   <HostId> L5/7vFC9nA/gZ5C16NctBs6QODLVZVTA8l/mY4GHdyPO2A47/oylkftRIkKEzOvRP6yDIM50eng= </HostId>
 </Error>
```

Infrastructure - Remaining Work

- Finalize **authentication** procedure
  - Embed authentication in each Lambda?
  - Create new “Authentication” Lambda?
- **Deploy** remaining components
  - Checkers Engine
  - CV
- Consolidate to single deployment pipeline from **main** branch

Timeline

Demo 1

Load pictures from gallery

Access device camera

Plan out Checkers Engine

3 separate pages on FrontEnd

AWS Research

Demo 2 Accomplishments

- Manual board creation on front end
- Unified board representation across modules
- Ability to detect piece color, location on board
- CheckersEngine logic to show valid moves
- Local authentication as Proof of Concept
- AWS Architecture setup with endpoints
- Gitlab -> Lambda CI/CD pipeline
- Image upload using pre-signed URLs

TODO in Demo 3

Testing for Backend Engine

Serverless authentication

Playable game on frontend with hints, AI gameplay

More Robust CV

Questions?