Apprentice Node.js Backend Assignment

The Task:
Build a "Chess as a Service" API

Specifically, please create a REST endpoint for each of the following use cases, keeping in mind that for this project, we only want to deal with moving/updating pawns (and *none* of the other pieces).

1. Create a new chess game
- If a game is created successfully, return a 200, along with a representation of the initial game-state that a client can use to create their UI
- The response JSON can be anything of your choosing
- Assume that clients are dumb, and that they rely on the API for the starting position.

2. Fetch a game's current state
- How you represent the game-state and send it back to the clients is up to you.
- A client should be able to take this game-state and build their UI, as well as know if white or black needs to move next.

3. Fetch *potential*, valid spaces to which a piece *could* move from its current position
- This endpoint can be used to determine where any piece – white or black – can possibly move from its current position.
- It does *not* need to take into account whose turn it is since no state is being saved.
- The response JSON can be anything of your choosing.
- The response should indicate if a potential move will result in them capturing an opposing piece.
- **Important:** Only implement this for pawns. *For the purposes of this project, non-pawn moves should result in an error code of your choosing.*

4. Update the game-state with a new move:
- This endpoint will be used by clients to attempt to make a move.
- The response JSON can be anything of your choosing.
- Assume that the API must check to make sure each move is valid and is made in the proper order (white, then black, then white, etc.)
- A 200 from this endpoint indicates that the move was valid and the game-state has been updated successfully.
- **Important:** Only implement this for pawns. *For the purposes of this project, non-pawn moves should result in an error code of your choosing.*

5. Fetch the history of all moves in a game
- *Do not* implement this endpoint's logic, but make it available for clients to hit.
- When fully implemented, this would return the history of moves made by both sides.
- For now, please return an error code of your choosing.

Limiting the Scope of the Project:

- For the purposes of this question, assume that 1. Many chess games can be created by many users, but 2. Each chess game is played by one person, against him/her-self, in a single browser window/app. In other words, do *not* worry about attempting to broadcast updates to other players and supporting a multi-user, live-time experience.
- To reiterate: Only worry about moving/updating ***pawns*** for endpoints 3 and 4. *Do not implement logic for endpoints 3 and 4 for any other pieces*.
- Because you are only moving/updating pawns, do *not* worry about detecting checkmate or stalemate, or promoting a pawn.

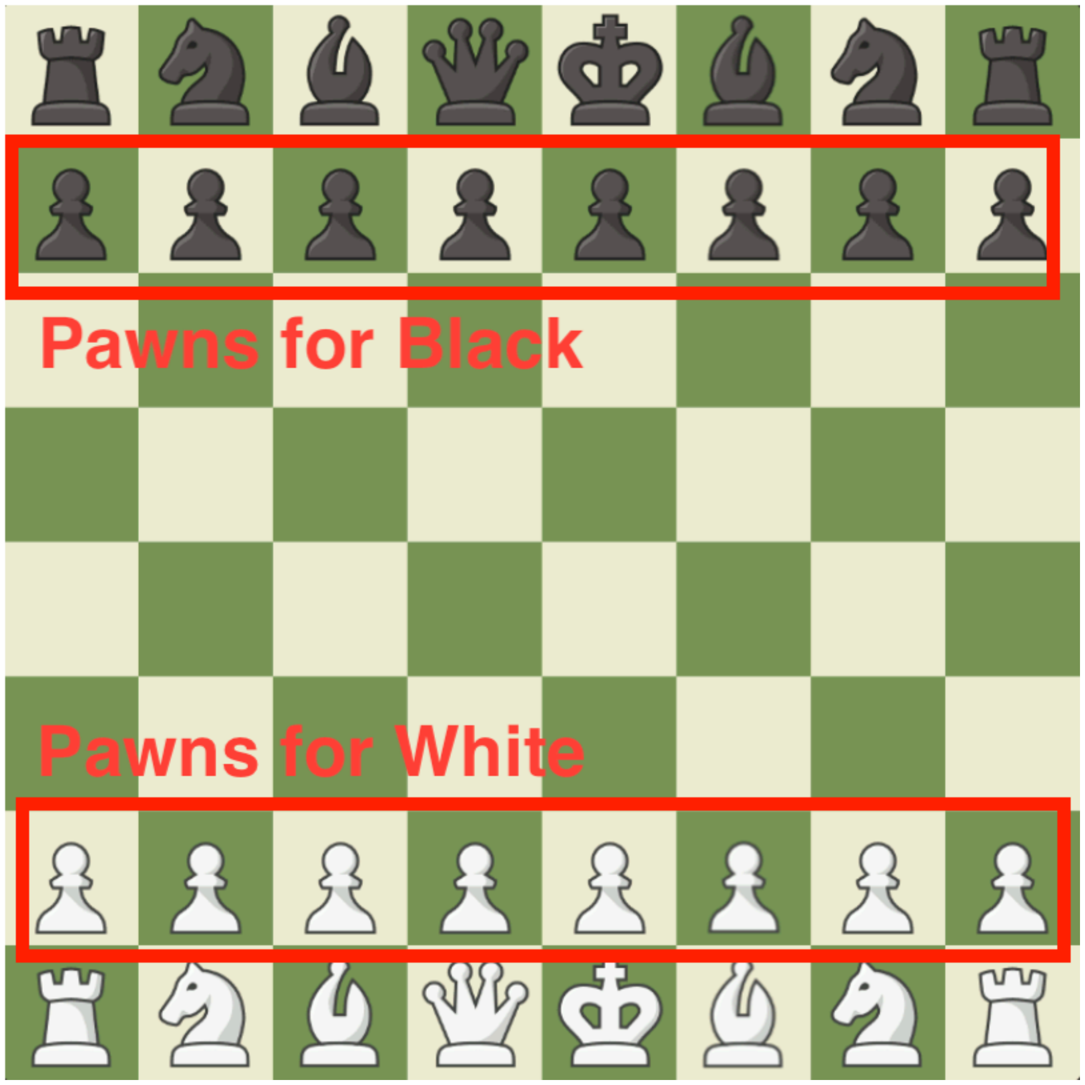
What we look for:

- First and foremost, a functionally complete REST API that conforms to normal standards and best practices.
- Proper architecture that allows for scalability, maintainability, and a reasonable degree of flexibility regarding future, additional requirements.
- An awareness of performance. When applicable, indicate when an operation may be prohibitively expensive and in need of future optimization.
- Clean and organized code that can be read and understood by others. Style counts!
- Be prepared to demonstrate what you've built and talk about all parts of it fluently. Be able to explain your architecture and design choices. Be comfortable talking about the pros and cons of your approach vs. other approaches you may have considered.
- We're not looking for a single, correct answer; rather, we want to see thoughtful decisions.

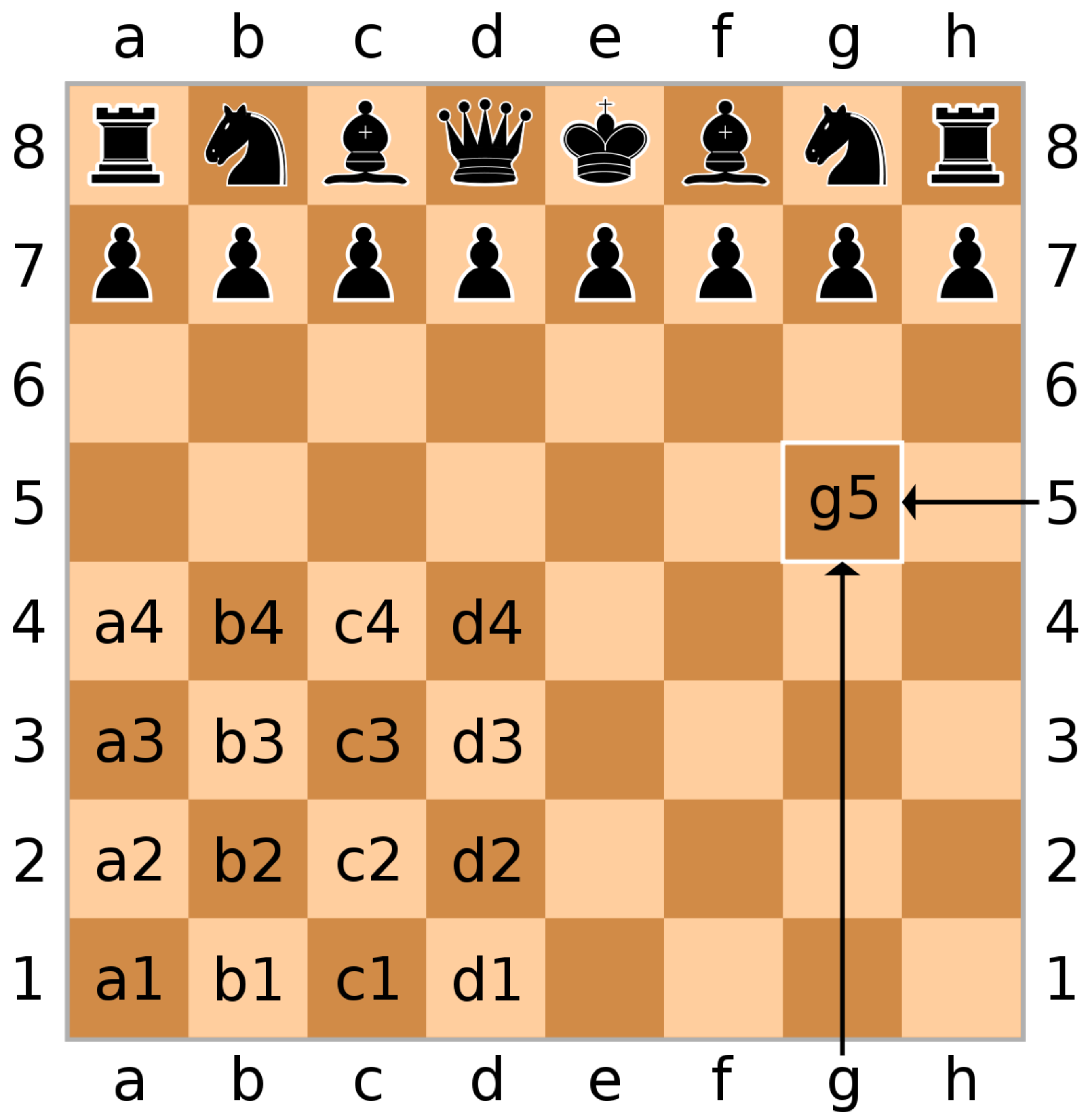
Important!

- Please feel free to use Express, Sails, or any other well known Node.js framework that you feel comfortable using.
- Please use MongoDB as your primary datastore, with an ORM of your choosing.
- Assume that we can test your project by spinning up your code in our local environment.
- Because there is no UI, assume that we will interact with your code via some set of curl commands (which are dependent on how you decide to implement the endpoints).
- Please do not use 3rd-party libraries unless absolutely necessary. If you do need to use a non-standard library, please indicate why you needed it.
- All work should be yours! An occasional StackOverflow (or an equivalent website) search is, of course, permitted; but please keep these to a reasonable minimum. If you absolutely must use content from StackOverflow (et al.), please cite the exact URL that you used. The goal of this is that the work you provide is your work, not someone else's – use your judgement.

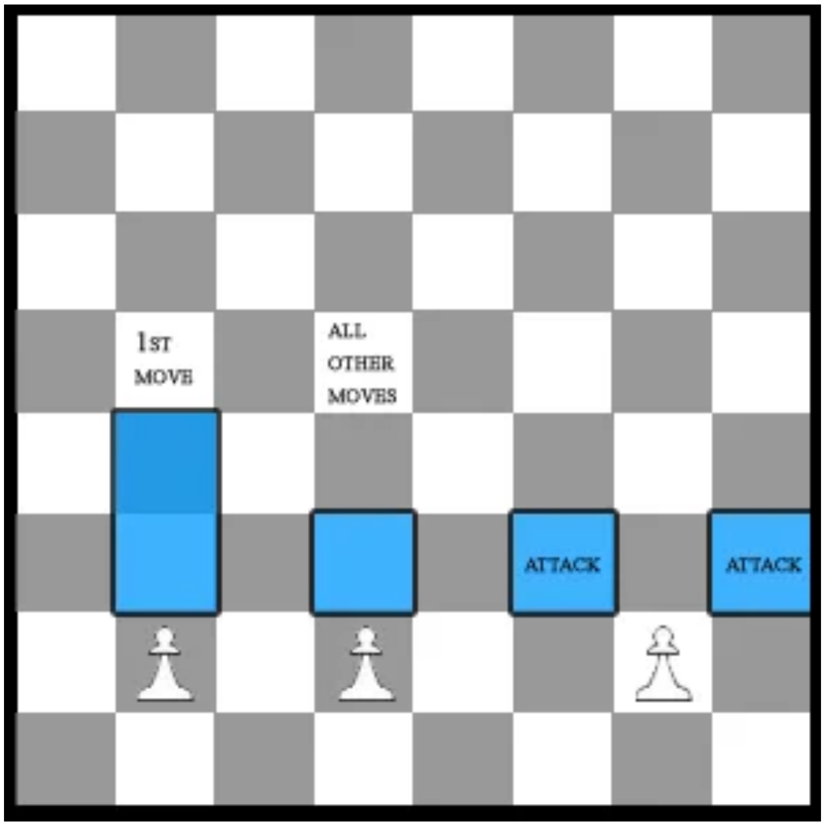
The Starting Position (White moves first)



The Standard Grid System (a1, b1, c1, etc.)



How Pawns Move



Usually, pawns move one square forward each move.

But, a pawn can advance either one *or* two squares on its *first* move.

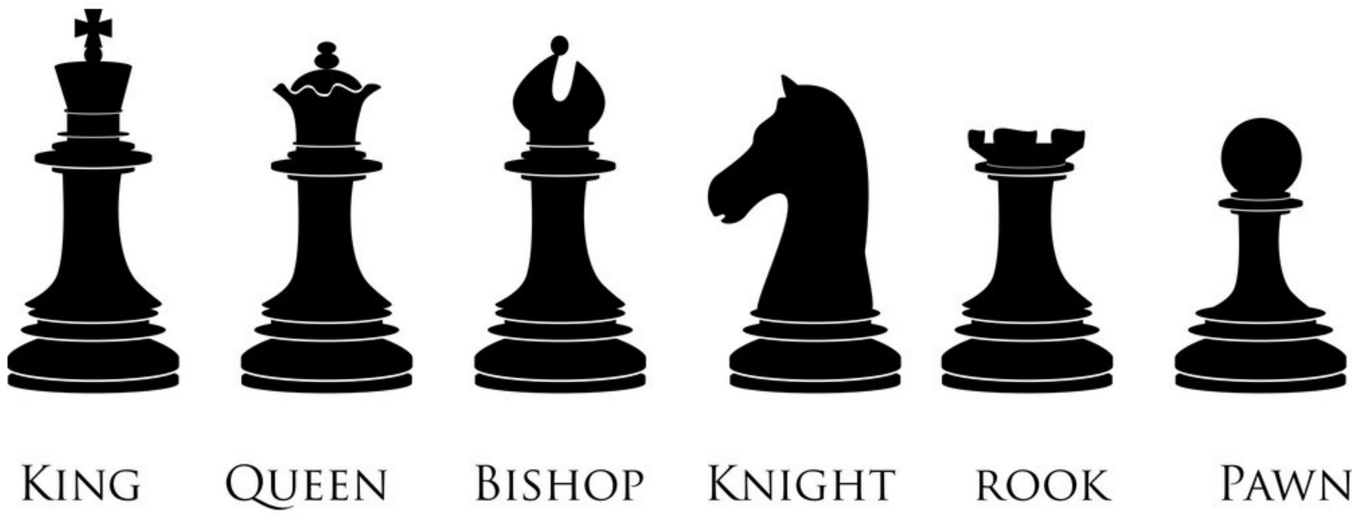
Additionally, pawns don't capture the same way that they move. Pawns can only capture an opposing piece that is one square *diagonally* ahead. (See the “attack” squares above)

Pawns are also the only piece that can only go forward.

https://www.chess.com/lessons/how-to-move-the-pieces/the-pawn

The Other Pieces

https://www.chess.com/terms/chess-pieces



Abbreviations:

King = **K**
Queen = **Q**
Bishop = **B**
Knight = **N**
Rook = **R**
Pawn = **P**