The goal of this challenge is to build an API that will manage (add, update, list, delete) players with some skills and select the best players with the desired position/skill.

The players will need to have:

- name
- position
- list of skills

The available positions for players are:

- 'defender'
- 'midfielder'
- 'forward'

The skill will have:

- skill name
- value

The available skills for a player are:

- 'defense'
- 'attack'
- 'speed'
- 'strength'
- 'stamina'

The player needs to have at least one skill, but it does not need to have values for all available skills. A valid player in JSON format is:

```json
{
  "name": "player name",
  "position": "midfielder",
  "playerSkills": [
    {
      "skill": "defense",
      "value": 60
    },
    {
      "skill": "speed",
      "value": 80
    }
  ]
}
```

You are also encouraged to write your own test cases under tests directory.

Basic Config:

The app needs to be served at http://localhost:3000 and the API requests should be available at http://localhost:3000/api/.

In case of errors, the body should return the correct error message following this pattern:

It is important that the error message contains the field that is invalid (position field in this case) and the invalid value used in the request (midfielder1 in this case).

If the field is inside an array, the error message should show the field inside it, for example if the skill of the player (inside playerSkills array) has an invalid value, the error should contain the "skill" field and the invalid value for that skill.

The solution should return only the first error found. If the request to create the player has invalid values for position and skill fields, the solution should return only the message for one of those fields. The validation rules do not need to follow any specific order.

Create Player:

The app will need to support player creation on http://localhost:3000/api/player.

The player info will be sent in the body in JSON format:

```
{
   "name": "player name 2",
   "position": "midfielder",
   "playerSkills": [
      {
         "skill": "attack",
         "value": 60
      },
      {
         "skill": "speed",
         "value": 80
      }
   ]
}
```

The expected result from this endpoint is the created player. The result can also have additional fields, like id's created for the player and the skills ex:

```
{
   "id": 1,
   "name": "player name 2",
   "position": "midfielder",
   "playerSkills": [
      {
         "id": 1,
         "skill": "attack",
         "value": 60,
         "playerId": 1
      },
      {
         "id": 2,
         "skill": "speed",
            "value": 80,
            "playerId": 1
        }
    ]
}
```

Update Player:

The app will need to support player update on http://localhost:3000/api/player/{playerId} where {playerId} is the id of the player that is being updated.

The player info will be sent in the body in JSON format:

```
{
    "name": "player name updated",
    "position": "midfielder",
    "playerSkills": [
        {
            "skill": "strength",
            "value": 40
        },
        {
            "skill": "stamina",
            "value": 30
        }
    ]
}
```

The expected result from this endpoint is the updated player. The result can also have additional fields, like id's created for the player and the skills ex:

```
{
    "id": 1,
    "name": "player name updated",
    "position": "midfielder",
    "playerSkills": [
        {
            "id": 3,
            "skill": "strength",
            "value": 40,
            "playerId": 1
        },
        {
            "id": 4,
            "skill": "stamina",
            "value": 30,
            "playerId": 1
        }
    ]
}
```

Delete Player:

The app will need to support player deletion on http://localhost:3000/api/player/{playerId} where {playerId} is the id of the player that is being deleted.

Important!
The endpoint to delete the player should be protected using Bearer token in the Authorization Header. The value of the Authorization Header should be:
Bearer
SkFabTZibXE1aE14ckpQUUxHc2dnQ2RzdlFRTTM2NFE2cGI4d3RQNjZmdEFITmdBQkE
=

List of Players:

The app will need to support an endpoint to list the available players on http://localhost:3000/api/player/.

The expected result from this endpoint is the list of players in the database, for example:

```
[
   {
      "id": 1,
      "name": "player name 1",
      "position": "defender",
      "playerSkills": [
         {
            "id": 1,
            "skill": "defense",
            "value": 60,
            "playerId": 1
         },
         {
            "id": 2,
            "skill": "speed",
            "value": 80,
            "playerId": 1
         }
      ]
   },
   {
      "id": 2,
      "name": "player name 2",
      "position": "midfielder",
      "playerSkills": [
         {
            "id": 3,
            "skill": "attack",
            "value": 20,
            "playerId": 2
         },
         {
            "id": 4,
                "skill": "speed",
                "value": 70,
                "playerId": 2
            }
        ]
    }
]
```

Team Selection:

The app will need to support an endpoint to select the players available based on some parameters sent on the request. The endpoint to select the best team should be http://localhost:3000/api/team/process.

The team requirements will be sent to the body in JSON format:

```
[
    {
      "position": "midfielder",
      "mainSkill": "speed",
      "numberOfPlayers": 1
    },
    {
      "position": "defender",
       "mainSkill": "strength",
       "numberOfPlayers": 2
    }
 ]
```

The expected result from this endpoint is the best list of players from the database according to the requirements ex:

```
[
   {
      "name": "player name 2",
      "position": "midfielder",
      "playerSkills": [
         {
            "skill": "speed",
            "value": 90
         }
      ]
   },
   {
      "name": "player name 3",
      "position": "defender",
      "playerSkills": [
         {
            "skill": "strength",
            "value": 50
         },
         {
            "skill": "stamina",
            "value": 2
         }
      ]
   },
   {
      "name": "player name 4",
      "position": "defender",
      "playerSkills": [
         {
            "skill": "strength",
            "value": 37
         }
      ]
   }
];
```

Rules To Select The Best Team:

1. Given a position and the skill desired for that position, the app should be able to find the best player in the database with that skill and position. If there are more than one player with the highest skill value, the solution can select any of those players.

2. The same skill can be used in different positions. For example: you can send a request with a requirement for defender with the highest speed, and a midfielder with the highest speed in the same request.

3. The request should allow the same position and skill combination only once but should accept the same position multiple times. For example: you cannot send a request with two requirements for defender with the highest speed, but you can send a request with a requirement for defender with the highest speed and a defender with the highest strength. In the example above the same player cannot be used multiple times.

4. If there are no players in the database with the desired skill, the app should find the highest skill value for any players in the selected position. For example, if in the database we have 3 defenders with these skills:
   - player 1 has {speed: 90}
   - player 2 has {strength: 20}
   - player 3 has {stamina: 95}

   And the requirements ask for a defender with defense skill, the app should select player 3, because there are no defenders with defense skill, and the defender with highest skill value is player 3. The same rule should be applied if the player has multiple skills, so if we have in the database the following players:
   - player 1 has {stamina: 90, speed: 100, strength: 20}
   - player 2 has {stamina: 80, speed: 80, strength: 80}

- player 3 has {stamina: 95, speed 90, strength: 50}

  And the requirements specify a defender with defense skill, the app should select player 1, because it is the player with highest skill: speed 100.

5. The app should always fill the number of required players with the correct position. For example, if the requirement is for 2 defenders, the app should find the best 2 defenders with the desired skill and use rule number 4 if there are no available defenders with the desired skill.

6. The app should return an error if there are no available players in the required position. For example, if the request requires 2 defenders and there is only one defender in the database, the app should return an error with the message: “Insufficient number of players for position: defender”. This rule should only be applied for positions. The skill requirement should follow the rules described in point 4.