Software Engineering Sprint 4

Nurgun Rafizade, Mohamed Ahmed Shaaban

April 27, 2021

Introduction

This sprint we received a codebase with properly working Signup/Login page and the frontend part for the the Player Screen. Thus, we decided to add backend functionality for adding, updating and retrieving game details.

Back-end

For the initial setup and deployment, please take a look at the README. We have focused our efforts on the backend functionality for this sprint, so the main testing should be done at the backend server. Once a registered Instructor creates a game with game ID, the Player can add, update and retrieve game details on the backend. Note that the game ID needs to be changed manually in the main.py as of yet.

We have added a new GAME DETAILS table in our database with following properties:

```
CREATE TABLE Game_Details(
    game_ID VARCHAR(40),
    position VARCHAR(15),
    player_ID INTEGER,
    week INTEGER,
    Incoming_Demand INTEGER,
    Incoming_Shipment INTEGER,
    Total_Inventory INTEGER,
    cost INTEGER,
    PRIMARY KEY (game_ID, position),
    FOREIGN KEY (player_ID)
        REFERENCES Player(player_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    FOREIGN KEY (game_ID)
        REFERENCES Game(game_ID)
        ON DELETE CASCADE
        ON UPDATE CASCADE

);
```

We have added backend functionality for the Player to be able to add game details:

```
 def add_gameDet(self, game_id, player_ID, position,week,\
        Incoming_Demand,Incoming_Shipment, Total_Inventory, cost):

        """
            Add Games to Game table using the arguments passed
        """
        cur = self.conn.cursor()

        gameDets=self.get_gameDet()
        gameDetsids = []
        id = game_id + position
        for row in gameDets:
            gameDetsid = row[0] + row[1]
        if game_id not in gameDetsids:
            cur.execute(
                "INSERT INTO Game_Details(game_ID, player_ID, week,Incoming_Demand,
                Incoming_Shipment, Total_Inventory, position, cost)\
                     VALUES (?, ?,?,?,?,?,?,?);",
                     (game_id, player_ID,week,Incoming_Demand,Incoming_Shipment
                     ,Total_Inventory, position, cost))
        else:
            return False

        self.conn.commit()
        cur.close()
        return True
```

We have added backend functionality for the Player to be able to update game details:

```
    def update_gameDet(self, game_id, player_ID, position,week,\
        Incoming_Demand,Incoming_Shipment, Total_Inventory, cost):

        """
```

```
            update the games details table that already exists
        """
        cur = self.conn.cursor()

        cur.execute(
            "UPDATE Game_Details SET week = ?, Incoming_Demand = ?,
            Incoming_Shipment = ?, Total_Inventory = ?, cost = ?
            WHERE game_ID=? and position = ?\
                ",(week,Incoming_Demand,Incoming_Shipment,
                Total_Inventory, cost,game_id,position))



        self.conn.commit()
        cur.close()
        return True
```

We have also added backend functionality for the Player to be able to retrieve game details:

```
    def get_gameDet(self):
        """Get all columns from game table

        Returns:
            [[GameDet 1, GameDet2, ....]
        """
        data=[]
        cur = self.conn.cursor()
        query="SELECT * FROM Game_Details;"

        for row in cur.execute(query).fetchall():
            data.append(row)

        cur.close()
        return data
```

As we had view table functions before, we have also added backend functionality to view Game Details table:

```
    def show_gameDets():
    """
        Prints all the datas of Game table from the database
    """
    query=cur.execute("SELECT * FROM Game_Details;")
    for row in query.fetchall():
        print(row)

    return
```

For the player screen we added the backend components and basic calculations. Furthermore, in the main.py we have also added a section to update the game details table and connect its data to Player data.csv

```
@app.route('/playerscreen', methods=['POST'])
def playerscreen(name=None):
    """adding and managing player screen data"""

    data = request.get_json()

    """ for next sprint: connect the data of this function to playerscreen.js """

    game_id = "Q8Kcl" # put a game id that exists for the code to work
    player_id = 1
    position = "Factory"
    week = 1
    Incoming_Shipment = randint(0,50)
    Incoming_Demand = randint(0,50)
    data=con.get_gameDet()
    flag = False
    gamedet =[]
    total_inventory = 0
    cost = 0

    for x in data:
        # if the primary key existed, then don't add new data
        # instead, retrieve the data with the same primary key
        if (x[0]+x[1] == game_id + position):
            flag = True
            gamedet = x
            break

    if (not flag):
        # adding the data to the database
        total_inventory = Incoming_Shipment-Incoming_Demand
        if total_inventory>0:
            cost = total_inventory * 2
        else:
            cost = total_inventory * 2 * -1
        gamedet = [game_id, position, player_id, week, Incoming_Demand,
        Incoming_Shipment, total_inventory, cost]

        con.add_gameDet(game_id, player_id, position, week,Incoming_Demand,\
                Incoming_Shipment, total_inventory, cost)
    else:
        # updating data in the database
        total_inventory = gamedet[6] + Incoming_Shipment-Incoming_Demand
        if total_inventory>0:
            cost = total_inventory * 2
        else:
            cost = total_inventory * 2 * -1
        con.update_gameDet(game_id, player_id, position,
        gamedet[3]+1, Incoming_Demand,\
                Incoming_Shipment, total_inventory, cost)
        gamedet = [game_id, position, player_id, gamedet[3]+1,
        Incoming_Demand, Incoming_Shipment, total_inventory, cost]


    dic1 = {}
    dic1['game_id']= gamedet[0]
    dic1['position']= gamedet[1]
    dic1['player_id']=gamedet[2]
    dic1['week']=gamedet[3]
    dic1['Incoming_Demand']=gamedet[4]
    dic1['Incoming_Shipment']= gamedet[5]
    dic1['Total_Inventory']=gamedet[6]
    dic1['cost'] = gamedet[7]

    # putting all of the players game in a csv file
    # so the front end can use the data of multiple weeks
    if (not flag):
        with open('../frontend/src/components/pages/Player_data.csv', 'w') as f:
            w = csv.DictWriter(f, dic1.keys())
            w.writeheader()
            w.writerow(dic1)
    else:
        with open('../frontend/src/components/pages/Player_data.csv', 'a') as f:
            w = csv.DictWriter(f, dic1.keys())
            w.writerow(dic1)

    return {'succes':True}
```

Testing

We have added backend testing for adding/updating game details, we use automated function that uses random data of games and users and creates 4 positions for each game with different users and checks whether the datas were successfully added or not.

Front-end

For frontend we added a simple button that connects to the back-end of Player Screen with the "next week" alert:

```
    const playerscreen = () => {
    axios.post('http://0.0.0.0:8086/playerscreen',
      {
      })
      .then(response => {
        // console.log(response.data)

        var x = response.data

        console.log(x)
      })
      .catch(error => {
        console.log(error.response)
      });
      alert("next week!");
  }
```