Andrew Brown (ajb5384)

Bibartan Jha (bj8355)

Morgan Murrell (mmm6855)

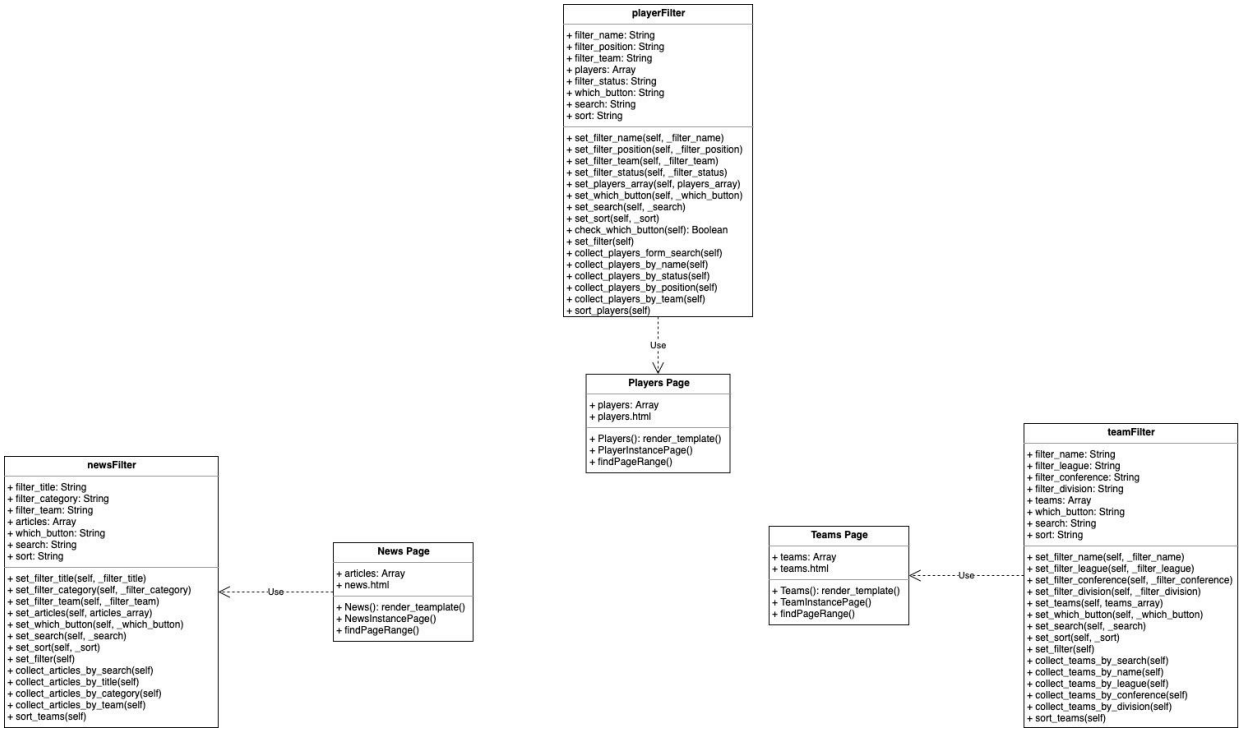
Alina Nguyen (amn2763)

Design Report

Information Hiding

For our design pattern, we implemented the concept of information hiding without our overall design scheme. Because we have been using the Python Flask framework in order to render different HTML pages on our website, we have discovered that it would be highly beneficial for future design changes if we modularize the code within our main Python file into different classes. There are several design changes that may change if we can continue to work on this project, however these changes will most likely be in our model pages (Players, Teams, and News since our model pages have several features for user interaction.

One of the biggest design changes on our model pages in the future will be the sorting, searching, and filtering schemes to the instances that users see on each model page. In the future, we can continue to find more information from Rest APIs about each model type, and thus there will be more attributes we can use to search, sort, and filter each model (i.e. we can find information about each team's mascot, and then we can sort the Teams instance based on mascot name). We have encapsulated this change by creating a separate class for each model page that implements all of the search, sort, and filter functionality. The class will also store all of the information needed for searching, sorting, and filtering our model pages. Storing the information inside of our classes will make adding more attributes to our model pages much easier, thus making our program much more robust. There are some disadvantages for our modularization approach. One big disadvantage is that our classes are not very abstract. We have created specific classes that correspond to specific model pages. To mitigate this issue, we made the classes have very similar functionality and logic with the searching, sorting, and filtering. While the information greatly differs between classes, we made sure that the logic to implement the filtering could possibly be shared between classes. The class diagram for the information hiding is shown below.



Information Hiding Class Diagram

Code Snippet for Information Hiding

```
class playerFilter:
    def __init__(self,players_array):
        self.players = players_array
        self.filter_name = ""
        self.filter_position = ""
        self.filter_team = ""
        self.filter_status = ""
        self.search = ""
        self.search_category = ""
        self.sort = ""

    def collect_players_from_search(self, search_category_request, search_request):
        self.search_category = search_category_request
        self.search = search_request
        if self.search_category != None and self.search_category != "None" and self.search_category != "Search Category" and self.search_category != "":
            if (self.search != None and self.search != "None" and self.search != ""):
                self.search = self.search.lower()
                players_temp = []
                for player in self.players:
                    player_attribute = player[self.search_category].lower()
                    if self.search_category == 'Status':
                        if self.search == player_attribute:
                            players_temp.append(player)
                    else:
                        if self.search in player_attribute:
                            players_temp.append(player)
                self.players = players_temp
```

```
class teamFilter:
    def __init__(self,teams_array):
        self.teams = teams_array
        self.filter_name = ""
        self.filter_league = ""
        self.filter_conference = ""
        self.filter_division = ""
        self.which_button = ""
        self.search = ""
        self.search_category = ""
        self.sort = ""

    def collect_teams_by_search(self, request_search_category, request_search):
        self.search_category = request_search_category
        self.search = request_search
        if self.search_category != None and self.search_category != "None" and self.search_category != "Search Category" and self.search_category != "":
            if self.search != None and self.search != "None" and self.search != "":
                self.search = self.search.lower()
                teams_2 = []
                for team in self.teams:
                    if self.search_category == 'Division':
                        if team['League'] == 'NBA':
                            team_attribute = team[self.search_category].lower()
                            if self.search in team_attribute:
                                teams_2.append(team)
                    else:
                        team_attribute = team[self.search_category].lower()
                        if self.search in team_attribute:
                            teams_2.append(team)
                self.teams = teams_2


class newsFilter:
    def __init__(self,articles_array):
        self.articles = articles_array
        self.filter_title = ""
        self.filter_category = ""
        self.filter_team = ""
        self.search = ""
        self.search_category = ""
        self.sort = ""

    def collect_articles_by_search(self, request_search_category, request_search):
        self.search_category = request_search_category
        self.search = request_search
        if self.search_category != None and self.search_category != "None" and self.search_category != "Search Category" and self.search_category != "":
            if self.search != None and self.search != "None" and self.search != "":
                self.search = self.search.lower()
                articles_temp = []
                for art in self.articles:
                    article_attribute = art[self.search_category].lower()
                    if self.search in article_attribute:
                        articles_temp.append(art)
                self.articles = articles_temp
```

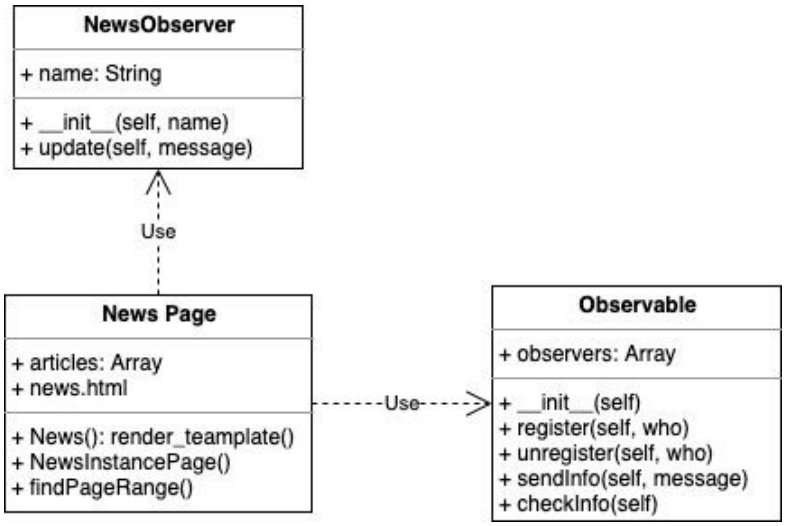
Design Patterns

For our project, we decided to apply the observer pattern. Since our website displays basketball news, we implemented an observer that will listen to our sports API if there are any newly added news articles. Our users will want to be kept up to date with the latest news about basketball games, teams, and players, so our observer will be notified if there is an update in the API so that we can add the information to our mongoDB database. Once it is added to our database, it will be available and displayed for our user whenever they request to view the news page. The code snippet and class diagram for our observer pattern is shown below.

Code Snippet of Observer Pattern:

```
class Observable:
    def __init__(self):
        self.observers = set()
        self.newsID = 0

        client = MongoClient("mongodb+srv://morganm:friedorboiled_@teame9db.kngdj.gc
        db = client["News"]
        col = db["NBA"]
        self.numberofNews = col.count()
        print(self.numberofNews)

        if col.count() == 40:
            # first time running
            getmostRecent = (col.find({}))[0]
            self.newsID = getmostRecent['NewsID']
        else:
            # not first time running
            getmostRecent = col.find_one({'NewsNumber':str(col.count()-1)})
            self.newsID = getmostRecent['NewsID']
        print(self.newsID)
        print('created observable\n')

    def register(self, who):
        self.observers.add(who)
        print('added observer\n')

    def unregister(self, who):
        self.observers.discard(who)
        print('removed observer\n')

    def sendInfo(self, message):
        for observer in self.observers:
            print('sending info\n')
            observer.update(message) # send dict of updated News to observer.

    def checkInfo(self):
        #checks API for new information
        getNewsID = checkAPI(self.newsID)
        print('\n\n\n\n')
        print(getNewsID)
        print(self.newsID)
        if self.newsID != getNewsID:
            # initialize dict to hold new news
```

observerP.py



observerP.py Class Diagram

Refactoring

For refactoring, one thing we did was reorganizing and compressing our various adding player/team pages for our comparisons chart into one adaptable html page. For example, we had an addcompareplayer.html confirmation page and an addcompareteam.html confirmation page, so we redesigned our code to have a single html page that could display the proper confirmation message despite the instance being added. Therefore, we were able to combine these 2 different adding routes into 1 route in order to condense our code and avoid repetition of code such as accessing the user's specific player and team choices in our database.

```
<div class ="container">

    <div class ="alert alert-success" style="...">
        <h2>{{name}} was added to your Comparisons Page.</h2>
    </div>


<div class="container p-5">
    <h4 style="text-align: center">ADDED {{type}} TO COMPARISONS TABLE!</h4>
    <!-- need to add form thing here with RESTful api but add later -->
    <div class="container" style="...">
        <h6>Go to the <a href="/comparison">Comparisons page</a> to check it out!</h6>
    </div>
</div></div>
{% endblock %}
```

addcompare.html

```
    # now find their document and prepare to add to favorite_player array
    if new == True:
        if type:
            if type == "player":
                if player in playerNames:
                    userscol.update_one({"_id": userUID}, {"$push": {"comparison_players": player}})

            if type == "team":
                if team in teamNames:
                    userscol.update_one({"_id": userUID}, {"$push": {"comparison_teams": team}})

            return render_template('addcompare.html', t=confirmation, name=name, type=type.upper(),
                                   log=currentuserpre["loggedin"])
```

main.py

We also combined the remove confirmation htmls for the Comparisons chart and also the any error response html into one template.

```
<div class ="container">

    <div class ="alert alert-danger" style="...">
        <h2>{{message}}</h2>
    </div>

<div class="container p-5">
    <h4 style="...">NO CHANGES WERE MADE</h4>
    <!-- need to add form thing here with RESTful api but add later -->
    <div class="container" style="text-align: center; width:500px;">
        <h6>Go back to <a href={{route}}>{{link}}</a> to view your {{pagename}}!</h6>
    </div>
</div>
</div>
{% endblock %}
```

adderror.html

```
link = "Favorite Players page"
route = "/favplayer"
pagename = "Favorite Players"
return render_template('adderror.html', t=confirmation, playername=player.title(),
                       log=currentuserpre["loggedin"], message=message, link=link, route=route, pagename=pagename)
```

```
message = 'You already have this team added to your Favorite Teams page.'
link = "Favorite Teams page"
route = "/favteam"
pagename = "Favorite Teams"
return render_template('adderror.html', t=confirmation, teamname=team.title(),
                       log=currentuserpre["loggedin"], message=message, link=link, route=route, pagename=pager
```

```
message = 'You already have this' + type + 'added to your Comparison Chart'
link = "Comparisons page"
route = "/comparison"
pagename = "Comparison charts"
return render_template('adderror.html', t=confirmation, name=name,
                       log=currentuserpre["loggedin"], message=message, link=link, route=route, pagename=pagename
```

main.py

Additionally, we shortened the table of random games featured on the homepage. It was initially manually formatted with several <tr> and <td>s, but we cleaned it up by using a ‘for loop’.

```
<table id="hp_table" class="center" style="width:80%; text-align: center">
    <caption>Highlights for 5 random games from the 2019 NBA regular season</caption>
    <tr>
        <th>Home</th>
        <th>Score</th>
        <th> vs </th>
        <th>Away</th>
        <th>Score</th>
    </tr>
    {% for x in range(5) %}
    <tr>
        <td>{{ scores[x][1] }}</td>
        <td>{{ scores[x][0] }}</td>
        <td> </td>
        <td>{{ scores[x][3] }}</td>
        <td>{{ scores[x][2] }}</td>
    </tr>
    {% endfor %}
</table>
```

index.html

Lastly, we shortened the return parameter lists by including the filter variables in a dictionary list for each model. Previously, we had returned each instance variable for each filter class separately which caused the list of return parameters to be quite lengthy.

```
playersDict = {
    'players': player_filter.players,
    'filter_name': player_filter.filter_name,
    'filter_position': player_filter.filter_position,
    'filter_team': player_filter.filter_team,
    'filter_status': player_filter.filter_status,
    'search': player_filter.search,
    'search_category': player_filter.search_category,
    'sort': player_filter.sort
}

lastpage_range = math.ceil(len(playersDict['players'])/6)
p = Paginator(playersDict['players'], 6) #6 entries per page
num_pages = p.num_pages
page = request.args.get('page', 1, type=int)
posts = p.get_page(page).object_list
p_range = findPageRange(page, 5, lastpage_range)
# return render_template('players.html', p_range=p_range, page=page, posts=posts, num_pages=num_pages, p
return render_template('players.html', playersDict = playersDict, p_range=p_range, page=page, posts=post
```

main.py

```
    team_filter.sort_teams(request.args.get('Sort'))

    teamsDict = {
        'teams' : team_filter.teams,
        'filter_name' : team_filter.filter_name,
        'filter_league' : team_filter.filter_league,
        'filter_conference' : team_filter.filter_conference,
        'filter_division' : team_filter.filter_division,
        'search' : team_filter.search,
        'search_category' : team_filter.search_category,
        'sort' : team_filter.sort
    }

    p = Paginator(teamsDict['teams'], 6) #6 entries per page
    num_pages = p.num_pages
```

main.py

```
newsDict = {
    'articles': news_filter.articles,
    'num_instances': len(news_filter.articles),
    'filter_title': news_filter.filter_title,
    'filter_category': news_filter.filter_category,
    'filter_team': news_filter.filter_team,
    'sort': news_filter.sort,
    'search': news_filter.search,
    'search_category': news_filter.search_category
}
```

main.py