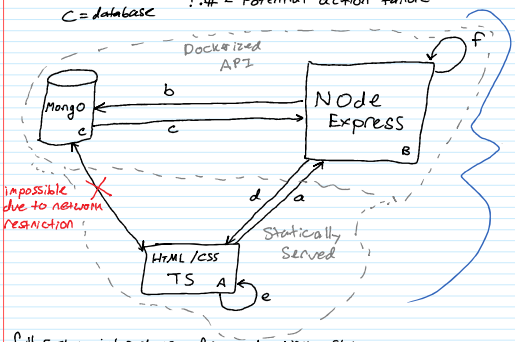
Top-Level System Architecture

Monday, September 16, 2019 11:53 AM

Legend: A = frontend, B = Backend, C = database; a, b, ... f = potential interactions; ?.# = Potential action failure



No choice in architecture here. Maintainence post-development could be an issue with frontend. Further no ODM is used on route b/c, making maintaining the backend a posible issue.

full System interactions for each User Story

CA - Create Account

1. a- User selects to create account and enters *{account_info}*
2. bcf- Backend ensures account name/email don't already exist
   a. b- Request users from DB
   b. c- Get response
   c. f- Search for re-use of email or display name
   d. (opt) d- if user already exists, report to frontend and exit.
3. f- Backend packages *{account_info}*
4. b- Backend stores new account in DB
5. d- Backend reports outcome to frontend.

LI - Log in

1. a- User selects to log in with *{login_info}*
2. bcf- Backend ensures account exists
   a. b- Request user with {login_info} from db
   b. c- Get response
   c. f- Check response for account
   d. (opt) if no account exists, report to frontend and exit.
3. f- Create login-token
4. d- Backend responds with login-token
5. e- Frontend adds token to browser

ATeTL - Add team to league

1. a- User selects to add team *{team}* to the league *{league}*
2. f- Backend creates a team object based *off {team}*
3. b- Backend requests league object for *{league}*
4. c- DB sends league object back to backend
5. f- Backend adds team object's ID to league object's teams list
6. b- Backend updates league object in DB
7. b- Backend sends new team object to DB
8. f- Backend sends email to *{team}.owner* informing them of new team
9. d- Backend reports successful creation to frontend

RTeFL - Remove team from league

1. a- Users selects to remove team {team} from league {league}
2. e- User confirms team name, and confirms to delete
3. b- backend requests league object from DB
4. c- database responds with league object
5. b- Backend requests to remove {team} from DB
6. f- Backend removes {team}.id from league object's teams
7. b- Backend sends updates league object to DB
8. d- Backend reports successful deletion to frontend

EL - Edit league

1. a- User selects to edit league {league} with {info}
2. b- Backend requests {league}'s object from DB
3. c- DB responds with {league}'s object
4. f- Backend updates league object with {info}
5. b- Backend requests to update league object with updated one
6. d- Backend reports successful update to frontend

VL - View League

1. a- Users selects to view {league}'s information
2. b- Backend requests {league}'s object from database
3. c- DB responds with league object
4. d- Backend responds with {league}'s object
5. e- Frontend formats data and displays

DL - Delete League

1. a- User selects to delete {league}
2. b- Backend requests {league}'s object from DB
3. c- DB responds with {league}'s object
4. f- Backend gets a list of {teams} that exist in {league}
5. b- Backend requests object for each team in {league}
6. c- DB responds with {teams} objects
7. b- Backend requests a list of all {matches} containing all matches from all {teams}
8. c- DB responds with {matches} objects
9. b- Backend requests objects for each tournament in {league}
10. c- DB responds with {tournaments} objects
11. b- Backend requests to delete all {matches} objects from DB
12. b- Backend requests to delete all {teams} objects from DB
13. b- Backend requests to delete all {tournaments} objects from DB
14. b- Backend requests to delete {league} from DB
15. d- Backend reports successful deletion to frontend

CTo - Create Tournament

1. a- User selects to create {tournament} for {league} with {teams} and {info}
2. b- Backend requests {league} object from DB
3. c- DB responds with {league}'s object
4. f- Backend creates {tournament} object with {teams} and {info}
5. f- Backend adds {tournament}.id to {league}.tournaments
6. b- Backend requests to update {league}'s object with updated object
7. b- Backend requests to add {tournament} to db
8. f- Backend sends emails to every {team}.*owner* for {team} in {teams}
9. d- Backend responds with successful creation to frontend

GTo - Generate Tournament

1. a- User selects to generate tournament for {tournament}
2. b- Backend requests {tournament}'s object from DB
3. c- DB responds with {tournaments}'s object
4. b- Backend requests a list of {teams} objects for each team in {tournament}.teams
5. c- DB responds with {teams} objects
6. f- Backend creates list of {matches} based off of ratings of teams
   a. Sort list of teams based off rating
   b. Let # of teams = n
   c. Create n-1 match objects
   d. Run match_alg on list of match objects
   e. If # of teams is odd: store highest rated team, and remove from list
   f. Iterating inwardly on *even* list of teams (ie, start at pos 0 and n, then 1, then n-1, etc): add each team encountered iteration to next match object [start at 0th match, add home then visitor, then move onto 1th match, etc)
   g. While adding to the match object, also add that match object's id to {team}.matches
   h. {matches} = list of match object generated
7. f- Backend adds {matches}.id IN ORDER to {tournament}.matches
8. b- Backend requests to add all of the match objects to DB
9. b- Backend requests to add tournament object to DB
10. b- Backend requests to update team objects with the new team objects (generated in 6b)
11. d- Backend responds with successful creation of tournament and with {tournament} object.
12. CONTINUE TO **VTo** [skip db request steps as {tournament} object returned here.]

VUTo - View Un-generated Tournament

1. a- User selects to view an un-generated {tournament}
   a. Frontend checks that user has permission to view this (see UI Page List)
2. b- Backend requests {tournament} object from db
3. c- Database responds with {tournament} object
4. d- Backend responds with {tournament} object
5. e- Frontend processes data and displays.

VTo - View (generated) tournament

1. a- User selects to view a (generated) {tournament}
2. b- Backend requests {tournament} object from db

```
matches = list(matches)
attach_to = len(l) - 1   ← final match
attach_from = len(l) - 2  ← start by attaching semi-final match 1 to final match.
num_attached = 0  ← used to track how many matches are attached to attach_to

while attach_from ≥ 0:

    matches[attach_from].victor_to = matches[attach_to]  ← link forward
    matches[attach_to].fill_from[num_attached] = matches[attach_from]  ← link backward.
    attach_from -= 1  ← decrement because this now attaches to attach_to
    num_attached += 1
    if num_attached == 2:  ← at most 2 matches attach to another match
        num_attached = 0  ← reset
        attach_to -= 1  ← start attaching to sub-matches
    end if
end while
```

match alg

```
matches = list(matches)

final_match = matches[len(matches) - 1]
processing = [[final_match]]
num_processed = 1
while num_processed ≤ len(matches):
    processing.prepend([])   ← prepend to front. DONT append to back
    for match in matches[i]:  ← look at each match in list of most recently processed matches
        num_processed += 1
```

display tournaments alg

2. b- Backend requests {tournament} object from db
3. c- Database responds with {tournament} object
4. d- Backend responds with {tournament} object
5. e- Frontend processes data and displays.

VTo - View (generated) tournament

1. a- User selects to view a (generated) {tournament}
2. b- Backend requests {tournament} object from db
3. c- Database responds with {tournament} object
4. d- Backend responds with {tournament} object
5. e- Frontend processes and displays information
    a. To display bracket, use the display_tournament algorithm.

display tournament alg

```
num_processed = 1
while num_processed ≤ len(matches):
    processing.Prepend([])
    for match in matches[:]:
        num_processed += 1
        if match.fill_from:
            for fill_match in match.fill_from:
                processing[0].append(fill_match)
            end for
        end if
    end for
end while
```

Prepend to front. DONT Append to back

O(n), one loop through list grabbing 2 items

Append to back DONT prepend to front

← look at each match in list of most recently processed matches

At this point, processing is a non-empty list with processing[0] being a list of all matches to be played in round 1, processing[1] all in round 2, etc.

ATofTe - Add team to tournament

1. a- User selects to add a {team} to {tournament} before generation
2. b- Backend requests {tournament} object from db
3. c- Database responds with {tournament} object
4. f- Add {team}.id to {tournament}.teams
5. b- Request to update {tournament} object in db
6. d- Report successful addition to frontend.

RTeFTo - Remove team from tournament

1. a- User selects to remove {team} from {tournament} before generation
2. b- Backend requests {tournament} object from db
3. c- Database responds with {tournament} object
4. f- Remove {team}.id from {tournament}.teams
5. b- Request to update {tournament} object in db
6. d- Report successful addition to frontend

ETo - Edit Tournament

Equivalent to EL - Edit League

DTo - Delete Tournament

Equivalent to DL - Delete league

CM - Create Match

1. a- User selects to create {match} with {teams}
2. f- Backend constructs match object from {match} and {teams}
3. b- Backend requests object for both {teams}
4. c- Backend responds with object for both {teams}
5. f- Backend adds {match} object to both {teams}
6. b- Backend requests to update both {teams} objects with updated versions
7. b- Backend requests to add match object
8. d- Backend reports successful creation to front end.

DM - Delete Match

1. a- User selects to delete {match}
2. b- Backend requests {team} objects for each team in {match}
3. f- Backend removes {match}.id from both {teams}
4. b- Backend requests to update both {team} objects in db
5. b- Backend requests to delete {match} object
6. d- Backend reports successful deletion to frontend

VM - View match

Equivalent to VL - View League

ETe - Edit team

1. a- User selects to edit {team} with {info}
2. b- Backend requests {team} object from db
3. c- Database responds with {team} object
4. b- Backend queries db for any teams with {info}.team_name to ensure no duplicate names
5. c- Database
6. f- Backend updates {team} object with {info}
7. b- Backend requests to update {team} object
8. d- Backend reports successful edit.

VTe - View team

Equivalent to VL - View League

LL - Leave League

1. e- Frontend ensures that {team} has scheduled matches (inc. tournament).
2. a- User selects for their {team} to be removed from {league}
3. b- Backend requests {league} object from db
4. c- Database responds with {league} object
5. b- Backend requests {team}'s object from db
6. c- Database responds with {team} object.
7. f- Remove {team}.id from {league}.teams
8. b- Backend requests to update {league} object
9. b- Backend requests to delete {team} object
10. d- Backend reports successful deletion

RR - Report Result

Case: Other team has not yet reported

1. e- Frontend ensures that {user} owns {team}
2. e- Frontend ensures {team} was in {match}
3. e- Frontend ensures that {match}.in_conflict is false
4. a- User selects to report {result} for {team} in {match}
5. b- Backend requests {match} object
6. c- Database responds with {match} object
7. b- Backend requests to update match object to reflect {result}
8. d- Backend reports successful report to frontend

Case: Other team has reported and no conflict occurs

1. e- Frontend ensures that {user} owns {team}
2. e- Frontend ensures {team} was in {match}
3. e- Frontend ensures that {match}.in_conflict is false
4. a- User selects to report {result} for {team} in {match}
5. b- Backend requests {match} object
6. b- Backend requests {team} object for winning team
7. c- Database responds with winning_team object
8. b- Backend requests {team} object for losing team
9. c- Database responds with losing_team object
10. f- Backend checks for conflict in {result} and current report
11. f- Backend set the match object to reflect outcome in {result}
12. f- Backed adds a win for winning_team
13. f- Backend adds a loss for the losing_team
14. f- Backend sets {match}.confirmed to true.
15. f- Backend calculates new rating for both teams and updates.
16. b- Backend requests to update both team objects
17. b- Backend requests to update match object
18. b- Backend requests {tournament} object for {match}.in_tournament (If non-existent, skip to step 22)
19. c- Database responds with {tournament} object
20. b- Backend requests {match} object for {match}.victor_to
21. c- Database responds with {match} object
22. f- Backend adds winning {team} to retrieved match object
23. b- Backend requests to update retrieved match object in database
24. d- Backend reports successful report to frontend

Case: Other team has reported and conflict occurs

1. e- Frontend ensures that {user} owns {team}
2. e- Frontend ensures {team} was in {match}
3. e- Frontend ensures that {match}.in_conflict is false
4. a- User selects to report {result} for {team} in {match}
5. b- Backend requests {match} object
6. b- Backend requests {team} object for winning team
7. c- Database responds with winning_team object
8. b- Backend requests {team} object for losing team
9. c- Database responds with losing_team object
10. f- Backend checks for conflict in {result} and current report
11. f- Backend recognizes conflict --- notification is sent to {league}.owner
12. f- Backend sets {match}.in_conflict to true
13. d- Backend reports unsuccessful report to frontend, and informs user of locked status.

SDOO - Settle Disagreement on Outcome

1. e- Frontend ensures that user is {league} owner
2. e- Frontend ensures that {match}.in_conflict is true
3. a- User selects to resolve conflict with {result}
4. b- Backend requests {match} object
5. b- Backend requests {team} object for winning team
6. c- Database responds with winning_team object
7. b- Backend requests {team} object for losing team
8. c- Database responds with losing_team object
9. f- Backend checks for conflict in {result} and current report
10. f- Backend set the match object to reflect outcome in {result}
11. f- Backed adds a win for winning_team
12. f- Backend adds a loss for the losing_team
13. f- Backend sets {match}.confirmed to true.
14. f- Backend calculates new rating for both teams and updates.
15. b- Backend requests to update both team objects
16. b- Backend requests to update match object
17. b- Backend requests {tournament} object for {match}.in_tournament (If non-existent, skip to step 22)
18. c- Database responds with {tournament} object
19. b- Backend requests {match} object for {match}.victor_to
20. c- Database responds with {match} object
21. f- Backend adds winning {team} to retrieved match object
22. b- Backend requests to update retrieved match object in database
23. d- Backend reports successful report to frontend